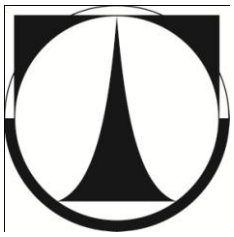
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií



BAKALÁŘSKÁ PRÁCE

Liberec 2012 **Adam Smolík**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646-Informační technologie
Studijní obor: Informační technologie

Záznam dat analyzátoru na přenosných zařízeních

Implementation of Analyser Data Storage Application on Mobile Devices

Bakalářská práce

Autor: **Adam Smolík**
Vedoucí: Ing. Jan Kraus
Konzultant: Ing. Pavel Štěpán

V Liberci 16.5.2012

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Abstrakt

V této práci je řešena problematika vytváření archivu z dat, stažených z multifunkčního měřícího a registračního analyzátoru kvality elektrického napětí. Výsledek práce je aplikace pro mobilní platformu Android, schopná stáhnout data z analyzátoru, správně je zapsat a zobrazit základní informace a grafy. Mobilní zařízení se k analyzátoru připojuje přes internet nebo wifi. Program byl vyvíjen v jazyce Java, upraveným a doplněným o knihovny pro práci s mobilním zařízením společností Google.

Klíčová slova: Android
Java
Mobilní zařízení
Zobrazování grafů
Analyzátor kvality el. napětí

Abstract

The work describes problems with creating an archive from a data downloaded from the multifunctional measuring and registrating analyzer of voltage quality. The result of work is an aplication for a mobile platform called Android. The aplication is able to download the data from the analyzer, correctly save it and show basic informations and graphs. The mobile device is connecting to analyzer by internet connection or wifi. The program was developed in the Java language with libraries allowing to work with an Android mobile.

Keywords: Android
Java
Mobile device
Drawing graphs
Analyzer of voltage quality

Obsah

1. Úvod ... 1

2. Seznámení s analyzátorem, komunikačním protokolem a formátem archivu ... 3

2.1 Analyzátor SMPQ/Simon PQ ... 3

2.2 Komunikační protokol *KMB Long* ... 3

2.3 Formát archivu ... 4

3. Android a použité technologie ... 7

3.1 Android ... 7

3.2 Použité technologie ... 7

4. Návrh řešení ... 10

4.1 Hlavní aktivita ... 11

4.2 Aktivita pro stažení CEA souboru ... 12

4.2.1 Připojení a dialog pro připojení ... 13

4.2.2 Nastavení intervalu ... 14

4.2.3 Stažení binárních archivů ... 15

4.2.4 Vytvoření CEA archivu ... 17

4.3 Aktivita pro rozbalení CEA souboru ... 18

4.4 Aktivita pro práci s binárními archivy ... 21

4.5 Zobrazení hlavního grafu ... 23

4.5.1 Aktivita pro zvolení obsahu ... 23

4.5.2 Uložiště dat hlavního archivu ... 24

4.5.3 Třída pro vykreslení grafu ... 26

4.5.4 Aktivita pro zobrazení dat hlavního archivu ... 27

4.6 Zobrazení electricity grafu ... 29

4.6.1 Dialog pro výběr obsahu ... 29

4.6.2 Úložiště dat electricity archivu ... 31

4.6.3 Třída pro vykreslení grafu ... 31

5. Vyhodnocení ... 33

5.1 Stahování CEA archivu ... 34

5.2 Rozbalování CEA archivu 35
5.3 Zobrazování binárních souborů 36
6. Závěr 37
Literatura 38

1. Úvod

Hlavním účelem této bakalářské práce bylo vytvořit mobilní aplikaci pro přenos dat z analyzátoru Simon PQ, který vyrábí firma KMB Systems. V průběhu realizace bylo na paměti, že uživatel s mobilním zařízením přistoupí k analyzátoru, který je připojený přes *ethernet* na bezdrátovou síť wifi. Pomocí tohoto programu se k dotyčnému zařízení připojí, získá z něj základní časové informace o záznamech, které analyzátor obsahuje, a rozhodne se pro stažení dat v časovém intervalu, jaký si uživatel sám určí. Program stažené záznamy správně uloží a zabalí do CEA archivu, který je čitelný pro program ENVIS, navržený firmou KMB Systems pro zobrazování a analýzu grafů.

Uživatel by také měl mít možnost stažený nebo jiným způsobem získaný CEA archiv rozbalit a vidět jeho binární soubory. Program by měl být schopný zobrazit informace o binárních souborech jak z hlediska souboru, tak z hlediska CEA souboru a následně vybrané binární archivy zobrazit ve formě grafu. Uživatel by měl mít dále možnost soubory mazat nebo přejmenovat.

Program byl vyvíjen pro mobilní zařízení běžící na operačním systému Android. Android je velice všestranný a rychle se rozvíjí, což nemusí být výhoda, ale naštěstí je zachována zpětná kompatibilita. Android má silnou programátorskou podporu, zajišťující dostupnost velkého množství vyřešených problémů. Programy pro Android se píší v Javě, která je jednoduchá a není problém najít spoustu vyřešených úloh. Platforma *Windows Phone* je zatím v plenkách a v době začátku vývoje programu byl tento systém nevhodný. Oproti tomu *iOS X* by vhodný pro vývoj byl, oproti Androidu se může pochlubit jednotností, co se týče softwaru i hardwaru. Systém *Applu* neexistuje v mnoha modifikacích jako tomu je u konkurence v podobě Googlu, kde si každý výrobce systém přizpůsobí dle svého gusta. Další z výhod, která stojí za zmínku, je určitě jednotnost poměru displejů u jednotlivých zařízení od firmy *Apple*, kdy zvětšení rozlišení displeje u nového typu je provedeno zdvojnásobením bodů displeje jak na šířku, tak na výšku. Nicméně během vývoje programu nebylo k dispozici zařízení *Applu*, proto jako cílová platforma byl zvolen Android.

Jako vývojový program byl vybrán *Eclipse* doporučovaný samotným *Googlem*. Aby bylo možné v programu *Eclipse* vyvíjet pro Android, bylo nutné nainstalovat *plugin* od společnosti *Google* a *Android SVK Tools* sloužící především pro emulaci

mobilního zařízení. Emulátor samotný nemusí sloužit pouze pro spuštění virtuálního zařízení, ale funguje zároveň jako most mezi reálným mobilním zařízením a programem *Eclipse*, díky čemuž je možné testovat aplikace přímo na reálném zařízení a to včetně *debuggeru* a *loggeru*. *Plugin,* umožňující vývoj pro Android v *Eclipse,* poskytuje veškerou základní funkčnost, jako například zobrazení a editaci obsahu externí i interní úložné paměti mobilního zařízení, nastavení statusu telefonu konkrétně stavu sítě (hledá, bez signálu, domácí síť, roaming nebo cizí síť), rychlost sítě (GPRS, EDGE, HSDPA atd.) a stav datového připojení a jeho zpoždění. Další zajímavé vlastnosti jsou simulace příchozího hovoru a textové zprávy a nastavení lokace pro GPS. Nastavení statusu, simulace hovoru, textové zprávy a lokace jsou dostupné pouze při použití virtuálního zařízení, pokud je program testován na reálném telefonu, tyto parametry jsou nastaveny dle dané sítě a pozice. Dále *plugin* umožňuje vytvořit fotku aktuálně zobrazeného obsahu na displeji. Více informací o programovacím jazyce Java viz. [20], více informací o programu *Eclipse* viz. [21].

2. Seznámení s analyzátorem, komunikačním protokolem a formátem archivu

2.1 Analyzátor SMPQ/Simon PQ

Jedná se o multifunkční panelový měřicí a registrační analyzátor kvality napětí. Je to model se zvýšenou přesností měření proudů, výkonů a elektrické energie. Dále umožňuje měření meziharmonických, míry vjemu blikání a týdenní vyhodnocení kvality napětí dle EN 50160.

2.2 Komunikační protokol *KMB Long*

Každé SMPQ zařízení je v základu vybaveno USB komunikací. Tento port je schopný emulovat virtuální sériovou komunikaci s počítačem hosta. USB port může být využit pro stažení dat, konfiguraci a kontrolu statusu pomocí komunikace *KMB Long*, který je podporován softwarovou sadou ENVIS. Volitelně může být zařízení vybaveno portem RS232 nebo RS485. Pomocí této sériové komunikace *KMB Long* nebo *Modbus RTU* poskytuje analyzátor všechny potřebné informace pro PC s ním propojené. Nejpokročilejší volba komunikace pro tento produkt je podpora *Ethernetu*. Tato volba umožňuje uživateli současně přístup k různým datům přes implementaci *Modbus RTU*, *KMB Long* protokolu založených na TCP/IP a skrytého webového serveru s aktuálními daty a konfigurací.

Komunikační kanál používá nastavení osmi datových bitů bez parity a s jedním stop bitem. Adresa a velikost datového toku může být nastavena. Komunikační protokol využívá filozofii *master-slave*. Po přijetí náležité zprávy nebo příkazu zařízení pošle zpátky relevantní odpověď. Všechny podporované zprávy musí mít formát rámce:

1. adresa analyzátoru (1 byte), hodnoty 0 a 255 jsou rezervovány
2. délka těla zprávy (2 byty)
3. typ zprávy (1 byte)
4. tělo zprávy, které záleží na typu zprávy
5. 16bit CRC.

Když analyzátor obdrží příkaz, pošle zpět odpověď. Byte obsahující typ zprávy v odpovědi obsahuje nulu, pokud nenastal žádný problém. V případě chyby je typ zprávy nastaven na 0x80 a je následován tělem zprávy o velikosti jednoho bytu

obsahujícího hodnotu chybového kódu. Všechny hodnoty jsou zakódovány v síťové zprávě jako *Big Endian*.

Zprávy jsou zakódovány podle typu. Čas a datum je vždy 64bitová hodnota zakódována jako počet milisekund od 1.1.2000. Samotné hodnoty záznamů jsou zakódovány pomocí transformovacího poměru, který definují multiplikátory MTN, MTNN, MTP a MTPN.

Pomocí správných zpráv lze z analyzátoru získat veškeré potřebné informace, jako jsou identifikace, aktuální data, konfigurace analyzátoru nebo uložená data. Knihovnu pro komunikaci s přístrojem SMPQ navrhl Pavel Novák. [17] [18]

2.3 Formát archivu

Stažená data z analyzátoru musí být správně zobrazena, proto je nutné je nejprve uložit do binárního souboru. Binární soubor má pevnou strukturu, která pokud není dodržena, způsobí nečitelnost. Při stahování dat z analyzátoru se vytváří několik binárních souborů, které jsou pomocí ZIP komprese zabaleny do jednoho souboru, kterému je přiřazena koncovka CEA. Binární soubory, které CEA archív obsahuje, mají stejnou strukturu a liší se jen v datech, které se do nich ukládají.

Analyzátor tyto soubory nevytváří, pouze poskytuje data k jejich sestavení. Sestavení samotné probíhá na počítači, který obsahuje potřebný software pro stažení dat.

Struktura binárního archivu je velmi zjednodušeně popsána v následující tabulce. Pro správné testování souboru je potřeba znát strukturu souboru naprosto přesně a to především, kdy jsou data uložena jako *Big Endian* a kdy jako *Small Endian*. [19]

No.	Položka	Velikost / formát	Hodnota
1.	[ARCHIVE_STRING]	7 bytů	'Archive'
2.	[VERSION]	1 byte	0x03
3.	[ARCHIVE_TYPE]	1 byte	xxx - popsáno dále
4.	[NUMBER_ARCHIVES]	u32	počet archívů uložených v tomto souboru
5.	[xxx_START_TIME]	u64	KMB-LONG čas prvního uloženého záznamu
6.	[xxx_LAST_TIME]	u64	KMB-LONG čas posledního uloženého záznamu
7.	[NOTEBOOK_LEN]	u16	délka NOTEBOOK ID řetězce
8.	[NOTEBOOK_BYTES]	[NOTEBOOK_LEN]	byty Notebook řetězce
	[ID_LEN]	u16	délka ID řetězce
	[ID_BYTES]	[ID_LEN]	byty ID řetězce
9.	[NCFG]	u32	počet konfiguračních setů v tomto souboru - vždy nastaveno na 0x01
10.	[CONFIG_1_LEN]	u16	první konfigurační set, počet bytů. KMB-LONG POUZE!!!
11.	[CONFIG_1_BYTES]	[CONFIG_1_LEN]	byty prvního konfiguračního setu. KMB-LONG ONLY!!!
12.	...		
13.	[CONFIG_NCFG_LEN]	u16	poslední konfigurační set, počet bytů
14.	[CONFIG_NCFG_BYTES]	[CONFIG_NCFG_LEN]	byty posledního konfiguračního setu
15.	[ARCHIVES]		záznamy přijaté z analyzátoru

Tabulka č.1 Struktura binárního souboru [19]

1. Nejprve je zapsána hlavička souboru, obsahující řetězec „Archive“.
2. Hned po ní verze archivu, což je jeden byte, udávající jakým způsobem jsou data do archivu zapsána. Zde je popisována verze 3.
3. Následuje jeden byte obsahující typ archivu. Tato hodnota udává, zda li se jedná o MAIN = 0, SPROFILE = 1, MPROFILE = 2, LOG = 3, PQMAIN = 4, PQEVENTS = 5, ELMER = 6, PMAX = 7, PQOSCILOGRAM = 8, PQEVENTTREND = 9.
4. Následuje 32bitový *little endian*,dále pouze LE, *unsigned integer* obsahující počet záznamů uložených v tomto archivu.
5. Poté jsou zapsány dvě hodnoty, čas prvního záznamu jako 64bitový LE *unsigned integer*.
6. A čas posledního záznamu v tomto souboru jako 64bitový LE *unsigned integer*.
7. Následují informace o analyzátoru, délka názvu zařízení a měření, uložena jako 16bitový LE *unsigned integer*. Délka je implicitně nastavena na 64. Poté je byte po bytu zapsán název zařízení a název měření, jejichž celková délka odpovídá hodnotě zapsané před nimi.
8. Dále jsou zapsány podrobnější informace o analyzátoru. Nejprve jejich celková délka jako 16bitový LE *unsigned integer* a pak následují samostatné informace,

které jsou zapsány jako *big endian*, dále jen BE. Zapsané informace obsahují například typ zařízení, verzi hardwaru, verzi softwaru, adresu zařízení a další.

9. Následně jsou zapsané konfigy, které jsou velmi důležité a potřebné pro správné zobrazení hodnot uložených v archivu. Nejprve je zapsán 32bitový LE *unsigned integer* udávající celkový počet setů konfigů. Jeden set obsahuje vždy všechny konfigy.
10. Poté je zapsána celková velikost následujícího setu konfigů uložená jako 16bitový LE *unsigned integer*.
11. A konečně samotné konfigy, které jsou vždy uloženy v následujícím pořadí, *arc config, install config, config, PQ config, OUT config a elmer config*. Vždy nejprve počet bytů konfigu jako 16bitový BE *unsigned integer* následovaný samotnými byty konfigu.
12. Tím jsou zapsány základní informace o archivu a analyzátoru a mohou být zapsány samotné záznamy. Nejprve jsou zapsány dvě 32bitové LE *unsigned integer* hodnoty, počet záznamů s konkrétním konfigem a číslo konfigů. Poté je zapsán 16bitový LE *unsigned integer* udávající počet bytů jednoho záznamu. Následují samotné byty jednotlivých záznamů, které dodržují délku zapsanou před prvním záznamem.

3. Android a použité technologie

3.1 Android

Android je softwarový balíček pro mobilní zařízení, který zahrnuje operační systém, *middleware* a klíčové aplikace. Android se distribuuje s množstvím nativních aplikací zahrnující i emailový klient, SMS program, kalendář, mapy, internetový prohlížeč, správce kontaktů a další. Všechny aplikace jsou napsané v jazyce Java. Android je založen na Linuxu verze 2.6, který poskytuje systémové služby jako bezpečnost, správa paměti, správa procesů, síťový zásobník a ovladače. Jádro je zároveň abstraktní vrstva mezi hardwarem a zbytkem softwaru. [1]

Více informací o programování aplikací pro platformu Android naleznete v knížce [22].

3.2 Použité technologie

3.2.1 *Linear* layout

Linear layout je rozestavění, jež umísťuje *widgety*, které obsahuje, do jednoho řádku, nebo do jednoho sloupce. Směr řady může být nastaven voláním metody *setOrientation*. Také je možnost nastavit gravitaci, která určuje polohu *widgetů*, voláním *setGravity* nebo určit, že specifický *widget* naplní zbývající prostor v layoutu tím, že mu určíme větší váhu. Základní orientace je horizontální. [2]

3.2.2 *TextView*

TextView zobrazuje uživateli text a umožňuje text editovat, pokud to komponentě nastavíme. *TextView* je kompletní textový editor, naneštěstí základní konfigurace neumožňuje editaci. Pro editaci je vhodné použít potomka *EditText*, který konfiguruje *TextView* pro editaci. [3]

3.2.3 *GridView*

Widget zobrazuje položky v dvourozměrné mřížce s posuvníkem. Položky se do mřížky načítají z *ListAdapteru*, který je potomkem *BaseAdapteru*, od kterého dědí třída *HomeScreenShortcutAdapter*, sloužící pro vytvoření položek pro *GridView*. [4]

3.2.4 *Spinner*

Widget, který zobrazuje jednu položku, kterou je možno po rozkliknutí vybrat z listu položek. Položky se *widgetu* předávají pomocí adaptéru. [5]

3.2.5 *Progress* dialog

Dialog zobrazuje průběh nějaké operace. Může zobrazovat i nějakou zprávu. Dialog může být zrušitelný. [6]

3.2.6 *ListView*

Widget zobrazuje položky ve vertikálním posuvném seznamu. Položky se komponentě předávají pomocí instance objektu ListAdapter. [7]

3.2.7 *TimePicker* a *DatePicker*

Widget volí čas, v obou jak 24hodinový tak 12hodinový formát. Hodiny i minuty se kontrolují pomocí vertikálního posuvníku. Hodiny i minuty se dají zadat pomocí klávesnice. Ve 12hodinovém formátu lze přepínat mezi AM/PM. *Widget* pro zvolení data je podobný, uživatel si pouze volí dny měsíce a roky. [8] [9]

3.2.8 *Context*

Instance třídy *context* obsahuje globální informace o prostředí aplikace. Umožňuje přístup k aplikací specifikovaným zdrojům a třídám, spouštění aktivit nebo odeslání a přijímání *intentu*. [10]

3.2.9 *Intent*

Intent je abstraktní popis operace, která bude vykonána. Může být použit s metodou *startActivity* ke spuštění jiné aktivity. *Intent* poskytuje možnosti pro přenos informací mezi dvěma rozdílnými aktivitami. [11]

3.2.10 *Environment*

Environment umožňuje přístup k proměnným prostředí. Obsahuje metody pro získání cesty k různým systémovým složkám, například datová složka, složka externí paměťové karty, nebo i hlavní kořenová složka. Dále poskytuje metody pro zjištění stavu externí paměťové karty, například, zda je připojená, výjmutelná, nebo právo aplikace pro zápis na externí paměť. Dále obsahuje statické proměnné, které slouží

k porovnání získaného stavu. V aplikaci je použita metoda *Environment.getExternalStorageState* v kombinaci se statickou proměnnou *Environment.MEDIA_MOUNTED*, aby bylo zajištěno, že se aplikace nespustí bez externí paměťové karty. [12]

3.2.11 *Handler*

Handler umožňuje posílat zprávy o průběhu spustitelných objektů, které jsou pomocí něj spojeny s frontou zpráv ve vlákně, ve kterém byl vytvořen. Každá instance *handleru* spojuje jedno vlákno s jednou frontou. Při jeho definici je třeba nastavit, jak bude reagovat na příjem zprávy. Zprávy se identifikují pomocí proměnné *what*. Každá zpráva může nést různé datové typy včetně objektů.

K poslání zprávy slouží metody *sendMessage, sendMessageAtTime* a další. Vyzvednutí zprávy se provádí pomocí metody *handleMessage,* kterou se předává zpráva ke zpracování.

Handler nemusí sloužit pouze k přenosu zpráv, ale i k vykonání nějaké akce ve vláknu, ve kterém se nachází. K tomu mu slouží metody post, *postAtTime* či metoda *postDelayed*, která je v programu použita pro kontrolu připojení mobilního zařízení k analyzátoru. Není-li tomu tak *handler* jeho snahy o připojení po sedmi sekundách ukončí. [13]

3.2.12 *Calendar*

Calendar je abstraktní třída pro převod mezi *Date* objektem a *integer* hodnotami jako jsou rok, měsíc, den, hodina a další. Potomci třídy interpretují datum podle specifického kalendářového systému. [14]

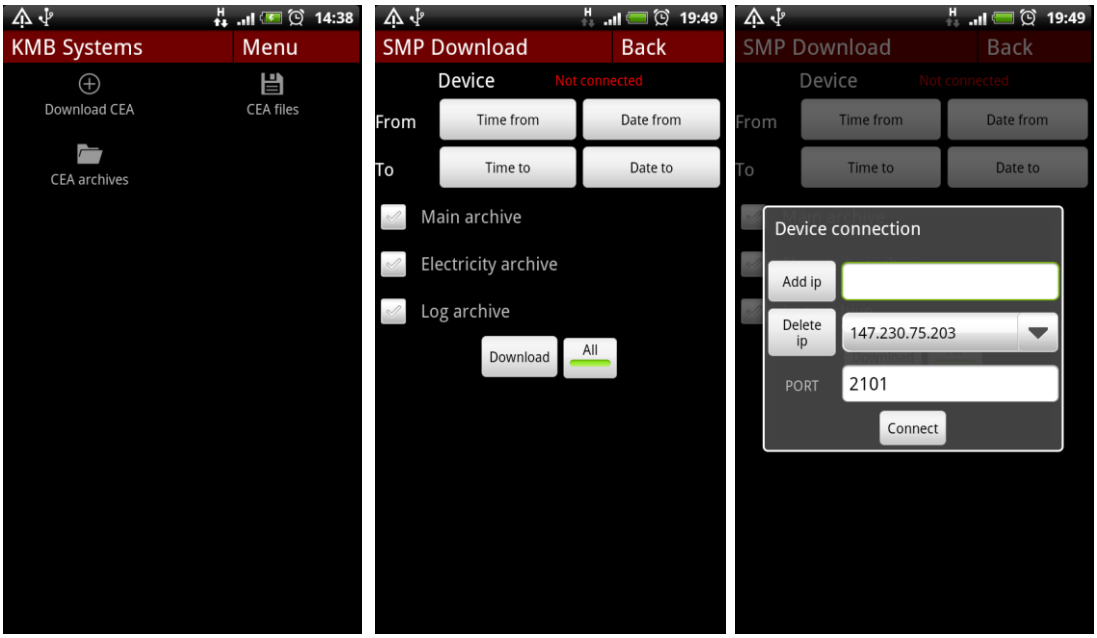
3.2.13 *ZipOutputStream*

ZipOutputStream je třída sloužící ke kompresi souborů do zip archivu. Zapisuje vstupní soubory do skrytého proudu. Výstupem je soubor odpovídající souboru zip. [15]

3.2.14 *Context* menu

Context menu se zobrazí po dlouhém kliknutí na položku. Tato položka musí být zaregistrována pomocí metody *registerForContextMenu*. Vytvoření menu se provádí pomocí metody *onCreateContextMenu*. [16]

4. Návrh řešení



Obr. č.1 a) Hlavní menu programu zobrazující ikony spustitelných aktivit. b) Stažení CEA archívu s nastavením času a konkrétních binárních archivů, které bude CEA soubor obsahovat. c) Dialog pro připojení k analyzátoru s možností přidat a odebrat adresu analyzátoru.

Program musí být snadno upravitelný, proto byl rozdělen na moduly, modul pro stažení CEA archivu, modul pro rozbalení CEA archivu a modul pro zobrazení informací o binárních archivech a zobrazení některých grafů. Tyto moduly jsou samostatné aktivity, které běží nezávisle na sobě a jejich změna nebo odebrání neohrozí funkčnost zbytku programu. Modul může spouštět další aktivitu. Pro spuštění aktivity jednotlivého modulu slouží hlavní aktivita, která je navržena jako křižovatka, na které se uživatel rozhoduje, jakou činnost chce konkrétně provádět.

4.1 Hlavní aktivita

Hlavní aktivita byla navržena s důrazem na jednoduchost a přehlednost. Aktivita sama o sobě slouží pouze jako rozcestník mezi ostatními aktivitami a její jedinou funkcí je kontrola dostupnosti paměťové karty, bez které ostatní aktivity nemohou fungovat.

Pro vytvoření grafiky byla použita kombinace *LinearLayoutu*, *TextView* a *GridView*. Grafika je navržena v XML. Vrchní lineární layout obsahuje lineární layout s hlavičkou a *grid view*, který obsahuje ikony aktivit. Orientace vrchního layoutu je nastavena vertikálně, zatímco layout obsahující dvě komponenty *TextView* má orientaci horizontální.

Při vytvoření aktivity se volá metoda *onCreate*, ve které je layout zobrazen pomocí metody *setContentView*. Komponenty *TextView* jsou navrženy jako statické a jejich obsah nelze programově měnit, zatímco komponenta *GridView*, sloužící pro zobrazení ikon pro spuštění dalších aktivit, se inicializuje až při vytvoření hlavní aktivity. Při spuštění aktivity se inicializuje instance třídy *GridView* pomocí ID, které jí bylo přiděleno v návrhu XML souboru. Této instanci je nutné dále přidělit adaptér, obsahující zdroj dat k zobrazení. Bylo rozhodnuto, že položka adaptéru by měla obsahovat jak text, tak i obrázek. K tomuto účelu byly navrženy dvě nové třídy, třída *menuResources*, která slouží jako úložný prostor obsahující objekt ikony typu *Drawable* a text typu *CharSequence*. Pole obsahující tři instance třídy *menuResources* je společně s *Contextem* předáno třídě *HomeScreenShortcutAdapter*, která se stará o vytvoření adaptéru pro *GridView*. Posledním krokem k vytvoření hlavní aktivity je přiřazení instanci třídy *Gridview listener*, který hlídá kliknutí na ikonu. Pro tuto událost se vytváří instance třídy *OnItemClickListener*, u které je přepsána abstraktní metoda *OnItemClick*, obsahující pozici prvku, na který uživatel klikne. Pomocí pozice a *switche* je rozhodnuto, která aktivita se spustí. Aktivity se spouští pomocí metody *startActivity*, které se předává instance třídy *Intent*. Nová aktivita se nespustí, pokud je nedostupné externí úložiště dat. Ke kontrole je použita funkce statické třídy *Environment*, konkrétně *getExternalStorageState*.

4.2 Aktivita pro stažení CEA souboru

Při návrhu této aktivity byl brán zřetel na jednu klíčovou vlastnost androidu, tzv. „*Painless Threading*“. Při vytvoření aktivity vzniká i UI vlákno, které se stará o zobrazování veškerých *widgetů* a obsluhu akcí, které nastanou jak působením uživatele, tak vlivem aplikace. Pokud je v tomto hlavním vlákně spuštěna nějaká časově náročná operace, vlákno se zasekne a a program přestane reagovat až do doby, kdy se zobrazí nechvalně proslulý „*aplication not responding*“(ARN) dialog.

Samostatná aktivita tedy pouze zobrazuje *widgety* sloužící k inicializaci připojení a dat ke stažení. Připojení a stažení dat potom probíhá v nových vláknech, které posílají hlavnímu vláknu informace o průběhu, dokončení stahování nebo o nějaké výjimce, která může nastat.

Pro návrh grafického rozhraní byl opět využit lineární layout s vertikální orientací, který obsahuje několik dalších lineárních layoutů s orientací pro změnu horizontální.

Vrchní layout obsahuje opět hlavičku, která se ale od hlavičky v hlavní aktivitě liší tím, že pravé komponentě *TextView* je přiřazen *OnClickListener*, který se stará o ukončení aktivity pomocí metody *finish* a o odpojení od analyzátoru, pokud je k němu mobilní telefon připojen.

Následuje layout pro připojení k přístroji, ten obsahuje tlačítko třídy *Button*, sloužící k zobrazení připojovacího dialogu, a Komponentu *TextView* zobrazující aktuální stav připojení.

Následují dva layouty obsahující informativní *TextView* a dvě tlačítka. Informativní text nám udává, zdali se jedná o počáteční čas nebo o konečný čas. Tlačítka poté slouží k vyvolání dialogu, který obsahuje buď *TimePicker* nebo *DatePicker* v závislosti na tom, o které se jedná. Tato tlačítka jsou po načtení aktivity neaktivní a není možné je stisknout. Aby bylo možné je použít, je nejprve potřeba se připojit k analyzátoru. Po připojení tlačítka stále nejsou aktivní, ale změní se jejich text tak, že vždy v prvním tlačítku je čas a ve druhém tlačítku je datum buď prvního nebo posledního záznamu v závislosti na tom, jestli jde o layout počátečního času nebo konečného času. Povolení stisknutí tlačítek je docíleno stisknutím *ToggleButtonu* v posledním layoutu s textem *All*, který slouží k rozhodnutí, zdali uživatel chce stáhnout všechny záznamy, které analyzátor zaznamenal, nebo si určit časový interval pro uložení záznamů.

Následují tři layouty, které obsahují vždy *CheckButton* a *TextView*. Jejich účel je jednoduchý, uživatel si kliknutím na ně zvolí, které archivy chce stáhnout.

Poslední layout obsahuje *Button* a *TogleButton*, tlačítko slouží k vytvoření vlákna pro stažení, přičemž než je vlákno vytvořeno, tak se testuje, zdali je zaškrtnutý alespoň jeden archiv. Přepínač určuje, zdali budou staženy všechny záznamy, nebo pouze interval. Tlačítko i přepínač jsou neaktivní a použít se dají až ve chvíli, kdy se uživatel úspěšně připojí k analyzátoru.

V této aktivitě se zobrazuje několik dialogů, konkrétně dialog pro připojení, dialog obsahující *DatePicker* a dialog s *TimePickerem*. Vytvoření dialogu se volá pomocí metody *showDialog*, které se předává *integer* určující, o jaký dialog se jedná. Pro tuto příležitost byly v aktivitě definovány statické *final integery*. Pro samotné vytvoření dialogu byly použity metody *onCreateDialog*, které se předává ID dialogu, a *onPrepareDialog*, do které vstupuje ID dialogu a objekt Dialog. Pro vytvoření dialogu lze použít pouze metodu *onCreateDialog*. Bylo však zjištěno, že je tento krok nevhodný, protože pokud byl už jednou dialog volán, znovu se nevytváří a uchovává si obsah, který v něm byl při jeho posledním zobrazení. To je vhodné u statických dialogů. V tomto případě, kdy je volán v programu například dialog s *DatePickerem* od dvou různých zdrojů a pokaždé s jinou hodnotou, je vhodnější použít kombinaci obou metod pro vytvoření dialogu, aby bylo zajištěno, že se vždy zobrazí s námi požadovaným obsahem, metoda *onCreateDialog* vrací prázdný dialog a metoda *onPrepareDialog* ho inicializuje, při opětovném zobrazení se již vyvolá pouze *onPrepareDialog*.

4.2.1 Připojení a dialog pro připojení

Připojení probíhá v samostatném vláknu, aby nebylo zablokováno UI vlákno. Pro připojení k přístroji je použita knihovna navržená Pavlem Novákem. K připojení slouží instance třídy *Connector*, která v konstruktoru přebírá řetězec IP adresy a *integer* port.

Dialog pro připojení je navržen pomocí XML. Návrh je opět vytvořen stylem vnořených lineárních layoutů. Dialog je navržen tak, aby bylo možné přidat i odebrat IP adresu. Přidané IP adresy jsou uloženy v binárním souboru, který se při prvním načtení dialogu přečte a jednotlivé adresy jsou uloženy do *ArrayListu* jako objekty typu *String*. Pro vybrání ip adresy byl použit *Spinner*, kterému je předán *ArrayAdapter*. Ten je vytvořen z *ArrayListu* obsahujícího *ipAdresy*. Kromě IP adres také uchovává informace

o vzhledu *Spinneru* a položek rozbaleného i nerozbaleného *Spinneru*. Celá tato procedura je zabalená v metodě *getIpList*, vracící *ArrayAdapter*.

Pro odebrání a přidání IP adresy byly navrženy metody *addIp, deleteIp*. Přidání je provedeno jednoduše připsáním položky do *ArrayListu,* aktualizován je i binární soubor. To proběhne pouze, pokud je IP adresa validní, k čemuž byla vytvořena metoda *ValidateIPAddress*, které se předá objekt typu *String*. Tento text je rozdělen podle teček do pole a jednotlivé prvky pole jsou testovány, zdali jde o číslo ve správném intervalu. Vymazání IP adresy je závislé na pozici zvolené ve *Spinneru*. Hledaný prvek je odebrán z předané pozice v *ArrayListu* a ten je opět zapsán do binárního souboru.

Pro připojení je navržena třída *inicializationThread*, která je potomkem třídy *Thread*. V konstruktoru se jí předávají instance třídy *Connector* a *Handler*. *Handler* je nástroj pro komunikaci mezi vlákny, pomocí něj se posílají zprávy o stavu inicializačního vlákna. Konkrétně to jsou veškeré výjimky, které mohou nastat, nebo zpráva o úspěšném připojení. Před samotným spuštěním vlákna je zobrazen *ProgressDialog*, který ukazuje zprávu o připojování k analyzátoru, a dále je vytvořena instance třídy *Runnable*, která má nastavené zpožděné spuštění po sedmi sekundách od spuštění inicializačního vlákna. Když se instance spustí, zkontroluje, jestli se vlákno úspěšně připojilo, a pokud tomu tak není, nechá zmizet dialog, zruší inicializační vlákno a vypíše zprávu o nepřipojení se k analyzátoru. Inicializační vlákno po spuštění provede připojení, a pokud je úspěšné, stáhne z analyzátoru data, z kterých se vytvoří instance třídy *SmpStatus*. Vytvoří zprávu, které předá tuto instanci a ID zprávy. K tomuto účelu byly opět vytvořené statické *final integery*, jejíž název odpovídá zprávě. Zpráva se pošle hlavnímu UI vláknu, kde ji *handler* zpracuje, zmizí připojovací dialog, změní text komponenty *TextView*, tím je uživatel informován o připojení a tlačítkům s časem změní nápis podle data prvního a posledního záznamu. Pokud vše neprobíhá správně a nastane výjimka, je vytvořena zpráva, které se předá jako objekt právě ta konkrétní výjimka a nastaví se správné ID. Zpráva je opět odeslána a uživateli se po zmizení připojovacího dialogu zobrazí text informující o nastalém problému.

4.2.2 Nastavení intervalu

Nastavení intervalu se provádí pomocí *DatePickeru* a *TimePickeru*, které jsou zobrazeny v dialogu. Těmto *widgetům* se při zobrazení nastavuje čas nebo datum pomocí vytvořených metod *setDate* a *setTime*. Metodám se předává *DatePicker* nebo *TimePicker* v závislosti na tom, o jakou metodu se jedná, a proměnná typu *long*,

ve které je uložen počet milisekund. V metodě se vytvoří instance třídy *Calendar*, z které jsou načteny potřebné proměnné a předány *TimePickeru* nebo *DatePickeru*.

Bylo třeba zajistit, aby uživatel nemohl zadat čas menší než minimální, větší než maximální a aby počáteční čas byl menší než koncový čas. Pro tuto příležitost byly navrženy čtyři funkce: *getStartTime*, *getStartDate*, *getLastTime*, *getLastDate*.

Metody jsou volány při stisknutí OK. Metodě *getStartTime* je předán *TimePicker*, aktuální startovní, konečný a první čas, který analyzátor zaznamenal. Provedou se testy a pokud nový čas úspěšně projde, tak metoda vrací proměnnou typu *long* představující počet milisekund, dialog zmizí a čas je aktualizován. Pokud testy neproběhnou v pořádku, v dialogu se nastaví hodnota, kterou měl při zobrazení, a uživatel je upozorněn, že zadal nesprávný čas. Stejný postup je navržen i u ostatních metod, jen jsou aktualizované testy, které se provádí, a proměnné, které jsou metodě předány.

4.2.3 Stažení binárních archivů

Stažení archivů se provádí v novém vlákně, aby nedošlo k zaseknutí UI vlákna. Stažení začne po stisknutí tlačítka *Download*. Před samotnou inicializací vlákna se zkontroluje, jestli uživatel zvolil alespoň jeden archiv ke stažení, zobrazí se *ProgressDialog* a načte se složka pro ukládání archivů. Složka se načítá pomocí metody *getExternalFilesDir*, které se předává řetězec s názvem složky. Pokud složka neexistuje, tak ji funkce vytvoří. Posledním krokem před inicializací vlákna je vymazání obsahu právě načtené složky. To je nezbytné, protože její obsah bude po ukončení zabalen do CEA archivu. Je žádoucí, aby archiv obsahoval pouze binární soubory, které vlákno stáhne. Mohlo by se totiž stát, že složka obsahuje archivy z předchozího stahování.

Před spuštěním vlákna je třeba jej vytvořit, konstruktoru se předává čas prvního a posledního záznamu v intervalu, který si uživatel zvolí, instance tříd *Connector*, *SmpStatus* a *Handler*, cesta k souboru do kterého se budou archivy ukládat v proměnné *String* a tři *booleany*, které určují archivy ke stažení.

Samotné stažení není nijak obtížné, knihovna pro připojení i stažení souborů funguje bez problémů. Bylo potřeba vyřešit otázku takzvaného *cyclingu* a chybějících záznamů. Analyzátor má totiž omezenou paměť, pokud naplní maximální počet záznamů, začne záznamy od začátku přepisovat. Pokud je poslán požadavek na první záznam a analyzátor cykluje, v odpovědi nebude první záznam, ale záznam, který

uložen je na první pozici. Index posledního záznamu v tomto případě je vypočítán jako součet celkového počtu záznamů a současného čísla posledního záznamu, které je v případě *cyclingu* záporné a slouží ke zjištění, zdali analyzátor cykluje. Vytvořilo se pole pozic, které je pomocí rotace otočeno tak, aby pozice prvků byly na adrese, na které se nacházejí v analyzátoru. Získání pozice prvku je potom provedeno jednoduše pomocí metody *getIndexOf*, které se předá požadovaná pozice.

Druhý problém, který bylo třeba vyřešit, jsou chybějící záznamy. Volí-li si uživatel interval pro stažení, jsou k dispozici pouze první a poslední čas, počet záznamů, NO posledního záznamu a časový interval, v kterém jsou záznamy pořizovány. Může se ale stát, že analyzátor bude určitý čas vypnutý a některé záznamy budou chybět. Pro tento účel byla vytvořena metoda *getPosition*, které se předává požadovaný čas jako *long*, typ archivu, maximální počet záznamů a NO posledního záznamu. Jako výsledek je vrácena pozice, na které je v analyzátoru uložen požadovaný čas. Jako vyhledávací algoritmus bylo zvoleno binární vyhledávání, které je v setříděném poli velmi rychlé. Algoritmus vždy stáhne záznam, načte čas a pokračuje podle principu vyhledávání. Pokud je záznam objeven, je vrácena jeho pozice, pokud se tak nestane, je vrácena pozice, na které se nachází záznam s nejbližším vyšším časem.

Na začátku se inicializují první a poslední pozice požadovaných archivů podle výše popsaného algoritmu a stahování může začít. Pro tento účel byla vytvořena metoda *writeArchive*, které se předává typ archivu, pozice prvního a posledního záznamu, číslo archivu, maximální počet záznamů a NO posledního záznamu. Pro zápis binárního souboru je použita instance třídy *Archive*. Metoda obsahuje několik *switchů* tak, aby byla univerzální pro všechny archivy, protože při zápisu je třeba rozlišit a správně vytvořit název archivu. Název se tvoří ze *stringu*, který identifikuje, že jde o archiv stažený mobilním zařízením, z názvu archivu, data a času, kdy byl soubor stáhnut. Zápis probíhá podle vzoru archivu verze tři. Pro zjednodušení této metody byly vytvořeny metody *writeIdentify* a *writeConfigs*, které přebírají instanci třídy *Archive*, zapisující data, které jsou pro všechny archivy stejné. Po dokončení stažení je pomocí *handleru* předána zpráva UI vláknu. Pokud nastane během stahování výjimka, je pomocí *handleru* taktéž předána UI vláknu, které ji zpracuje a informuje uživatele o nastalém problému.

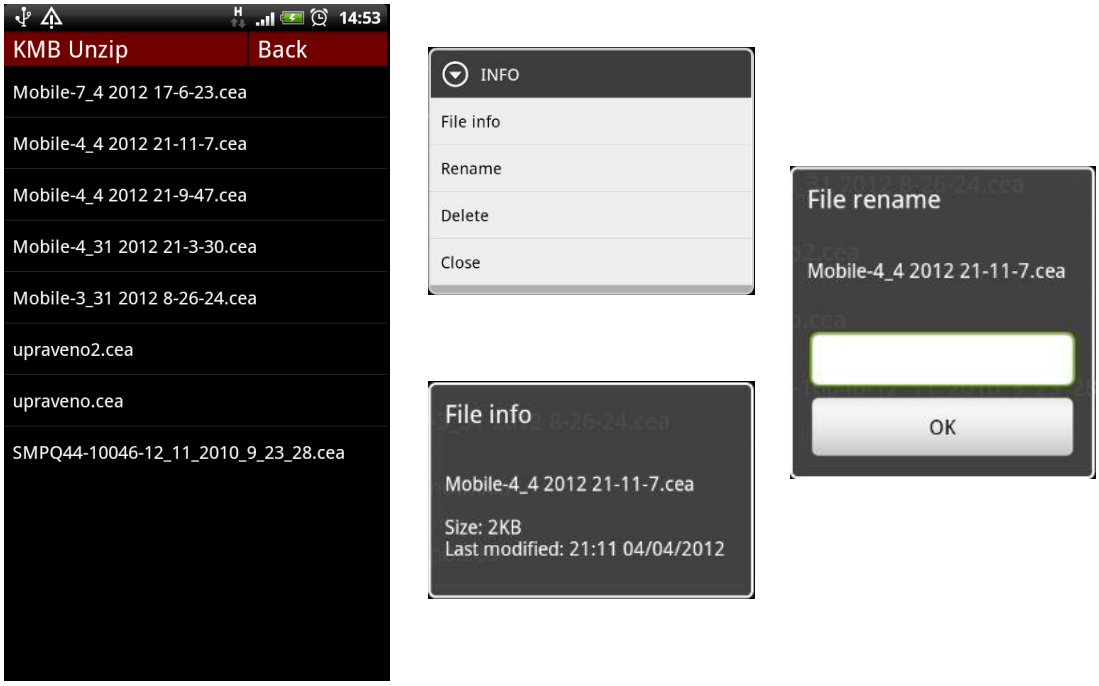
4.2.4 Vytvoření CEA archivu

CEA archiv je vstupní soubor do programu *Envis*. Pro jeho zabalení se používá komprese ZIP. Rozlišení od běžných archivů typu zip je potom v koncovce, kde zip je nahrazeno cea.

Komprese je časově náročná operace, proto je umístěna ve vlastním vlákně. Před vytvořením instance třídy zip je ukončeno spojení s analyzátorem, dialog zobrazující zprávu o stahování je nahrazen dialogem se zprávou o kompresi. Konstruktoru třídy zip se předává cesta s názvem CEA souboru, který bude vytvořen. Jeho název obsahuje informaci o tom, že byl vytvořen v mobilním zařízení, a čas, kdy byl vytvořen. Pro tuto příležitost je opět použita třída *Calendar*. Dále se kontruktoru předává složka obsahující binární archivy a *handler* pro komunikaci s UI vláknem.

Ve vláknu je vytvořena instance třídy *ZipOutputStream* a pole souborů pomocí metody *listFiles*. Zabalování probíhá v cyklu *for*, kde pro každý soubor v poli je vytvořena instance třídy *FileInputStream*, v zip souboru se vytvoří nový vstup metodou *putNextEntry* a následně jsou do ní zapsány všechny byty daného archivu. O úspěšném či neúspěšném archivování je hlavní vlákno informováno pomocí *handleru* a v závislosti na výsledku zmizí dialog a případně se vypíše chybová hláška.

4.3 Aktivita pro rozbalení CEA souboru



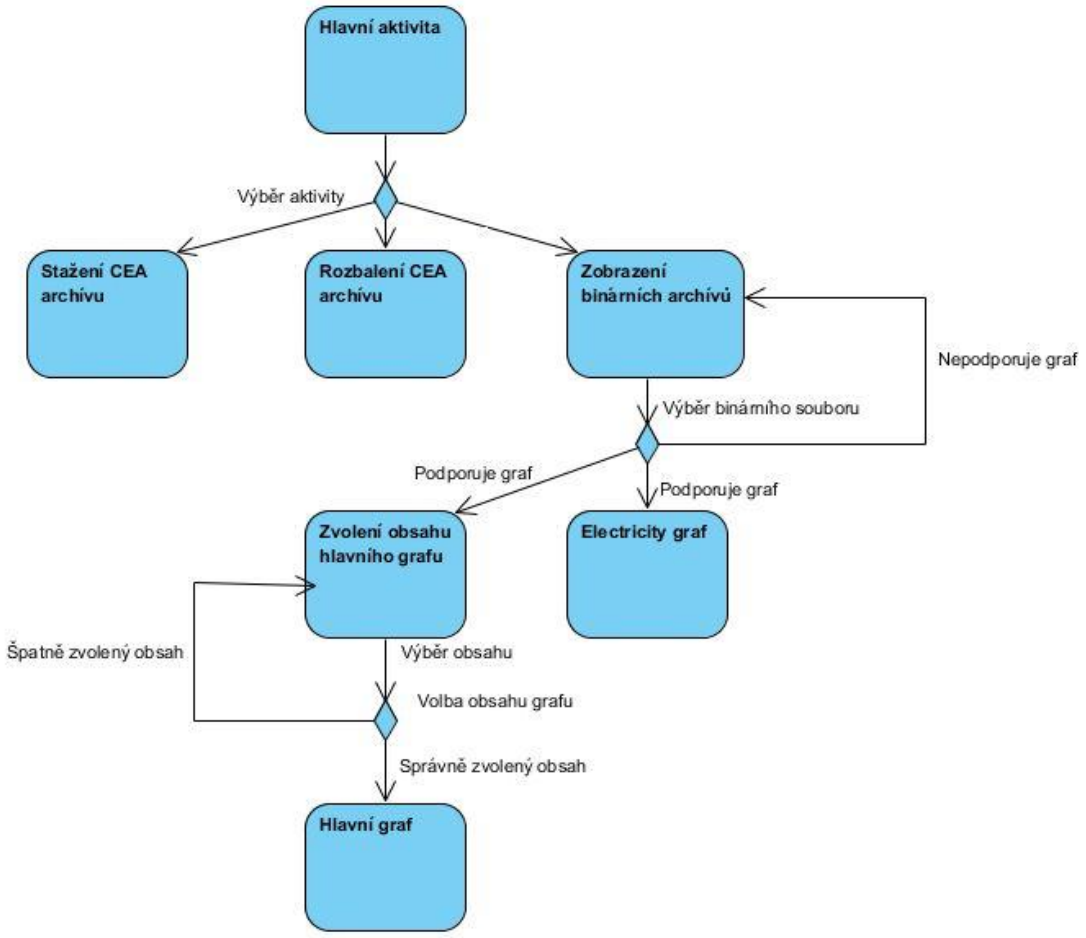
Obr. č.2 a) Aktivita zobrazující CEA archivy, po kliknutí je lze rozbalit. b) Dialog zobrazený po dlouhém kliku na archiv.
c) Dialog zobrazující základní informace o archivu. d) Dialog přejmenování archivu.

Program umí zobrazovat data a informace o binárních archivech. Proto musela být vytvořena aktivita, která CEA soubor zabalený pomocí zip komprese rozbalí. Kromě rozbalování umí aktivita také soubory mazat, přejmenovat, nebo zobrazit základní informace o archivu.

Aktivita pro rozbalení souboru je velice jednoduchá, jedná se o výpis názvů souborů, které jsou uloženy v příslušné složce na paměťové kartě mobilního zařízení.

K výpisu jednotlivých archivů byl navržen XML layout. Tento layout má klasickou hlavičku skládající se z dvou komponent *TextView*, kde jedna slouží pro návrat do hlavní aktivity. Komponenty jsou vnořeny do lineárního layoutu a ten je spolu s komponentou *ListView* obalen hlavním lineárním layoutem. Komponenta *ListView* umožňuje snadné zobrazování jednotlivých archivů.

Pro inicializaci výpisu archivů byla navržena metoda *initListView,* která se kromě volání na začátku volá ještě při smazání či přejmenování souboru. K načtení archivů je použita metoda *getExternalFilesDir*, která vrací soubor obsahující potřebné archivy. Následně je získáno pole řetězců s jejich názvy pomocí metody list. Pole je použito k vytvoření *ArrayAdapteru*, který je předán komponentě *ListView* k zobrazení. Posledním krokem je inicializace komponenty, aby se s ní dalo pracovat. To je vyřešeno



Obr. č.3 Struktura programu.

metodou *registerForContextMenu*, která *ListView* registruje pro vyvolání kontextového menu. Kontextové menu je vyvoláno dlouhým stiskem libovolné položky. Poté je už pouze nastaven *OnItemClickListener* pro samotné rozbalení archivu.

Rozbalení archivu je velmi náročná operace, a proto je vykonána v novém vlákně, aby nezasekla UI vlákno. Uživateli se před startem rozbalování zobrazí *progress* dialog, vytvoří se instance třídy *Handler*. Konstruktoru rozbalovacího vlákna se předá cesta k dotyčnému souboru, soubor, do kterého se archivy zkomprimují, a *handler*. Pro samotné rozbalování je použita třída *ZipFile* v kombinaci s výčtem a cyklem *for*. Problém byl řešen pomocí výčtu souborů obsažených v archivu, protože archiv může obsahovat soubory, které mají v názvu diakritiku, díky čemuž nefunguje metoda *getEntry* instance třídy *ZipFile*. Výčet archivů se inicializuje pomocí metody *entries*, která znaky s diakritikou nahradí neznámým znakem. Byla proto navržena metoda *replaceAccent*, které se předává název souboru a ten je po zpracování metodou vrácen se správným znakem bez diakritiky. Rozbalení poté probíhá jednoduchým čtením instance třídy *ZipEntry*, která je vytvořena získáním elementu z výčtu a následným zápisem bytů do binárního souboru. Po ukončení rozbalování podá vlákno zprávu

pomocí *handleru* a uživatel je informován o dokončení stahování zmizením *progress* dialogu.

V hlavním vláknu bylo třeba implementovat metodu *onResume,* protože pokud uživatel aplikaci shodí či zamkne mobilní zařízení, vlákno stále rozbaluje. Zpráva o dokončeném rozbalování se v tomto stavu do UI vlákna nedostane, protože je zmražené, a je zařazena do fronty. Z této fronty je zpráva získána pomocí metody *handleru obtainMessage*.

Pro vytvoření kontextového menu je přepsána metoda *onCreateContextMenu*. Metodě je předána instance třídy *ContextMenu*, které stačí nastavit text hlavičky a přidat jednotlivé položky.

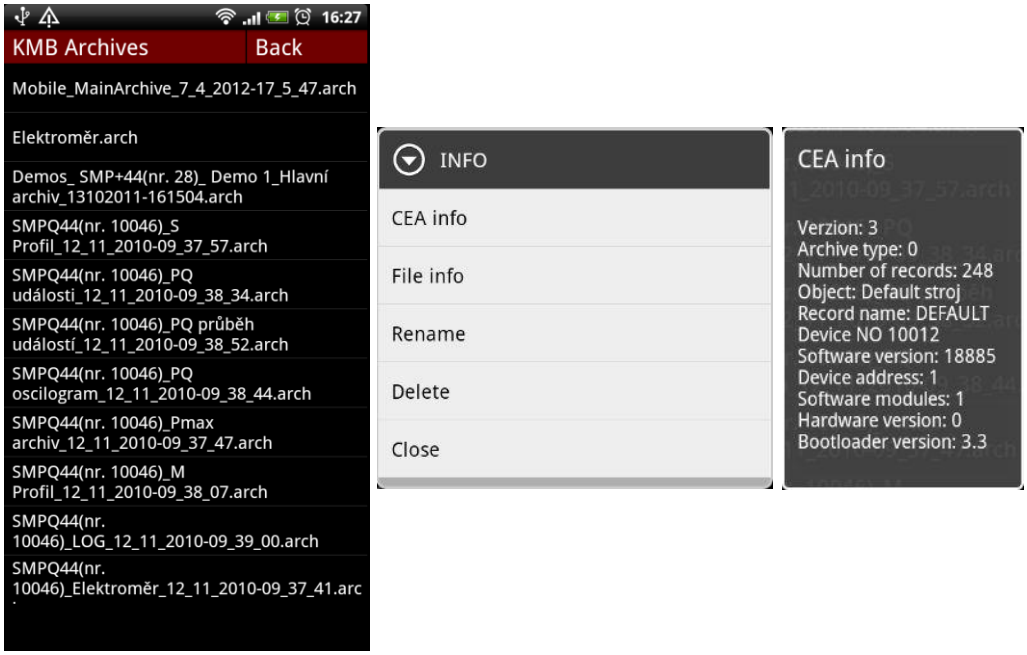
Kontextovému menu se nepřidává *listener*, ale je potřeba přepsat metodu *onContextItemSelected*, která přebírá instanci třídy *MenuItem*. V této metodě se o správně přiřazení operace stará *switch*.

Po kliknutí na položku *File info* se zobrazí jednoduchý dialog. Informuje uživatele o velikosti CEA archivu a časovém údaji poslední modifikace.

Pokud uživatel chce soubor smazat, vytvoří se instance třídy *File* a provede se smazání pomocí metody *delete*. Pokud je smazání úspěšné, obnoví se *ListView* a uživatel je informován o úspěšném smazání, v opačném případě je informován o neúspěšném smazání.

Dialog pro přejmenování je velice jednoduchý. Zobrazí se původní název, komponenta *EditText,* pro zadání nového názvu, a tlačítko pro potvrzení změny. Po potvrzení je nejprve zkontrolováno, jestli uživatel zadal text. Pokud je *EditText* prázdný a probíhá snaha o přejmenování, je uživateli zobrazena zpráva informující ho, že nebyl zadán nový název. Pokud kontrola projde, je ověřeno, zdali se zadaný název již ve složce nenachází. V tomto případě je požadovaný soubor přejmenován, *ListView* obnoven a uživatel informován o úspěšné změně názvu. V opačném případě je uživateli zobrazen text informující o existenci jména.

4.4 Aktivita pro práci s binárními archivy



Obr. č.4 a) Výpis binárních archivů. b) Dialog, zobrazený po dlouhém kliku na archiv, slouží k další práci s ním. c) Dialog zobrazující podrobnější informace o CEA archivu.

Tato aktivita byla navržena jako rozcestník pro zobrazování grafů, mj. má podobnou funkčnost jako aktivita pro rozbalení CEA souboru, ale je navíc obohacena o výpis informací o binárním archivu. Z této aktivity se spouštějí další dvě, aktivita pro zobrazení grafu *electricity* archivu a aktivita pro zobrazení grafu *main* archivu.

Z obrázku č. 4 je zřejmé, že pro tuto aktivitu byl použit stejný XML layout jako u aktivity rozbalující CEA archiv. (viz. výše)

Opět byla navržena funkce *initListView*, která reaguje na smazání nebo změnu názvu archivu. V této metodě je inicializován *onItemClickListener*, který v tomto případě nespouští vlákno pro rozbalování zip archivu, nýbrž vytváří instanci třídy *Intent*, sloužící ke spuštění aktivity pro zobrazení grafu. Před samotným spuštěním aktivity je nutné ověřit, zdali zvolený binární archiv je podporován v zobrazování grafu. Rozbalený CEA soubor totiž neobsahuje pouze *main* a *electricity* archivy, ale i spoustu dalších, které se prozatím nedočkaly podpory. Pro tuto kontrolu byla navržena metoda *checkAvailability*, které se předává instance třídy *String*, obsahující cestu k dotyčnému souboru. V metodě se pomocí cesty k souboru vytvoří instance třídy *ArchiveReader*, která inicializuje metodou *initInfo* informace o binárním archivu. Poté je pomocí metody *getArchtype* získán příslušný typ a ten je porovnán s podporovanými archivy.

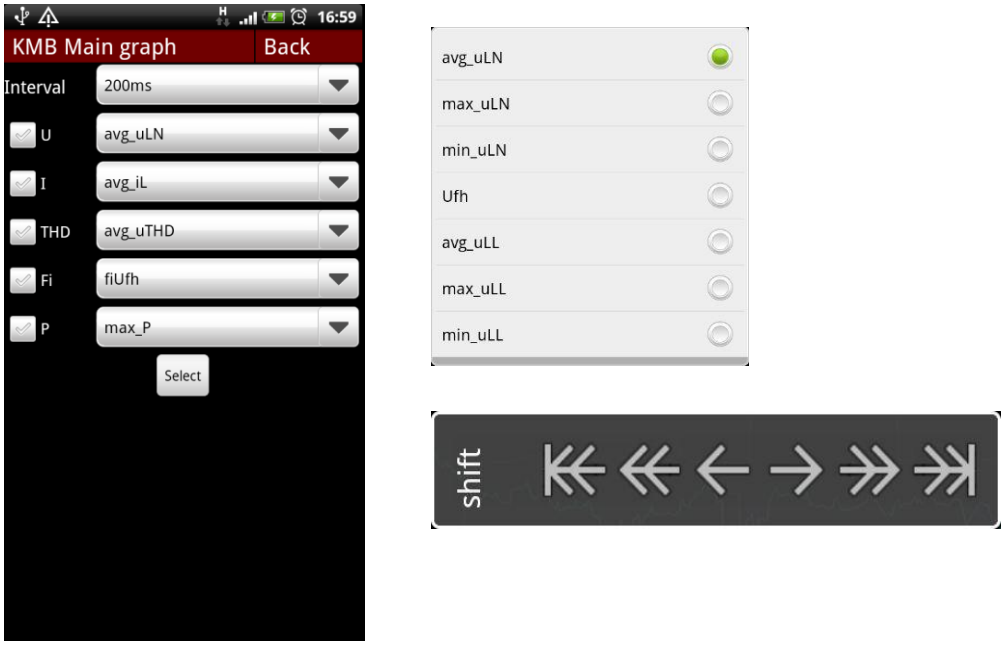
Pokud je podporován, je vrácen právě tento typ v podobě *integeru*, pokud není podporován, je vrácena hodnota -1.

Aktivita implementuje metodu *onCreateContextMenu* pro zobrazení menu sloužícího pro základní operace s binárními archivy.

Na výběr jsou pro uživatele CEA *info,* zobrazující informace týkající se binárního archivu z hlediska uložených dat a *File info*, které poskytuje základní informace o názvu, velikosti a datu poslední modifikace souboru. *Rename* a *delete* se užijí pro změnu názvu a smazání souboru. Pro obnovení komponenty *ListView,* po těchto dvou operacích, je použita metoda *initListView*. Poslední položka *close* slouží k uzavření dialogu.

Navíc oproti kontextovému menu z aktivity pro rozbalování CEA archivů přibyla položka pro zobrazení informací o binárním a archivu s názvem CEA *info*. Po kliknutí právě na tuto položku se vytvoří instance třídy *ArchiveReader*, které je předána příslušná cesta k binárnímu archivu. Pokud je jeho vytvoření úspěšné, začne se pomocí metod *showDialog*, *onCreateDialog* a *onPrepareDialog* vytvářet informativní dialog. Opět je použita kombinace metod *onCreateDialog* a *onPrepareDialog*, protože pokud by tomu nebylo tak a dialog by se vytvářel jen pomocí metody *onCreateDialog*, každé jeho další zobrazení by obsahovalo stejná data jako při prvním zobrazení. Dialog je totiž jednou vytvořen a při jeho druhém zobrazení už se metoda *onCreateDialog* nevolá a pouze se zobrazí. V metodě *onPrepareDialog* je implementován rozhodovací algoritmus *switch*, který rozhoduje na základě předaných informací, zdali se bude vytvářet dialog pro zobrazení informací o CEA archivu, o binárním souboru, nebo jestli se má zobrazit dialog pro přejmenování. V dialogu pro CEA informace je na instanci třídy *ArchiveReader* volána metoda *initInfo*, po jejíž volání se přečtou v binárním souboru základní informativní data. Tuto metodu lze volat i u souborů, které zatím nemají podporu zobrazování grafů, protože tato informační data jsou pro všechny archivy zapsána stejně. Dialog obsahuje *widget TextView*, do kterého jsou pomocí metody *append* zapsány údaje k zobrazení.

4.5 Zobrazení hlavního grafu



Obr. č.5 a) Výběr obsahu hlavního grafu. b) Obsah *Spinneru* zobrazující typ proměnné, kterou lze načíst. c) Dialog pro pohyb v grafu.

Hlavní graf je načítán z *main* archivu. Před samotným jeho zobrazením je spuštěna aktivita pro výběr hodnot grafu. Hlavní archiv obsahuje velké množství dat a program dokáže najednou zobrazovat maximálně dva druhy dat. Po vybrání jsou hodnoty načteny a následně zobrazeny.

4.5.1 Aktivita pro zvolení obsahu

Aktivita pro zvolení obsahu grafu byla navržena kvůli snížení paměťových nároků aplikace. Graf umí zobrazovat maximálně dvě dávky dat, a proto by bylo zbytečné načítat všechny, tzn. přes čtyřicet dávek, kde každá dávka obsahuje čtyři *integery,* a počet záznamů přesahuje sto tisíc. Pro relativně malou paměť mobilního zařízení tento fakt představuje velký problém. Byla proto navržena aktivita, ve které se volí data k zobrazení.

Layout je navržen tak, aby odpovídal standardnímu rozhraní aplikace s hlavičkou informující o zobrazené aktivitě a *TextView*, který má nastavený *listener* pro uzavření aktivity a návrat zpět. Celý layout je obalený v lineárním layoutu, který obsahuje další lineární layouty umístěné vertikálně. Po hlavičce následuje horizontální lineární layout, který obsahuje *spinner* určení ke zvolení intervalu, ve kterém se budou

záznamy načítat. Dále jsou pod sebou umístěny layouty obsahující *check box* a *spinner* v poloze horizontální. *Check box* slouží k potvrzení zobrazení dat a *spinner* k výběru konkrétních dat. Jako poslední je v layoutu umístěné potvrzovací tlačítko.

Po jeho stisknutí proběhne kontrola počtu zaškrtnutých *check boxů*, a pokud je kontrola úspěšná, vytvoří se instance třídy *Intent*, které se pomocí rozdělovacího algoritmu *switch* a metody *putExtra* přidělí identifikační název a číselná hodnota udávající pozici vybrané položky ve *spinneru*. Dále je načtena hodnota ze *spinneru* obsahující časový interval záznamů. Jako poslední proměnná je *intentu* předána cesta k souboru, kterou jsme této aktivitě předali z předchozí aktivity také pomocí *intentu*. K získání proměnné, kterou *intent* obsahuje, slouží metoda *getIntent*, vracející instanci třídy *Intent*, která byla vytvořena ke spuštění této aktivity.

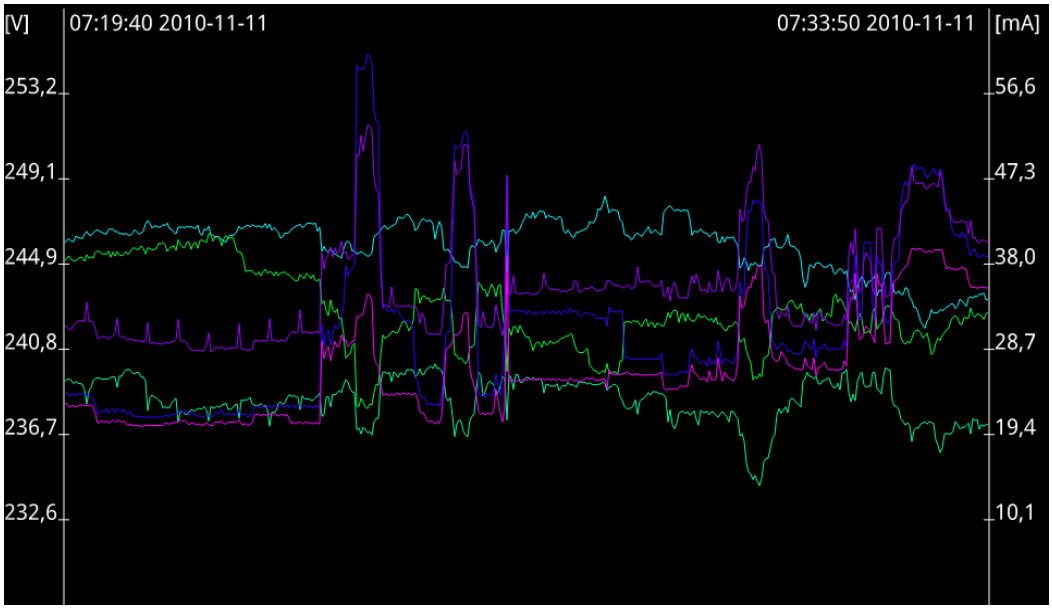
Při výběru konkrétních dat k zobrazení je potřeba použít *spinner*. K tomuto účelu byla vytvořena pro každý *spinner* zvlášť *array*, která je obsažena v XML souboru. *Spinner* slouží pouze k výběru konkrétních dat, a proto neimplementuje žádný *listener*. Aktivita obsahuje statické proměnné, které se používají k identifikaci těchto konkrétních zdrojů dat. Pro každý *spinner* jsou číslovány od nuly, tyto proměnné jsou využity v aktivitě grafu.

4.5.2 Uložiště dat hlavního archivu

Graf může být správně zobrazen tehdy, pokud jsou vytvořeny třídy, které slouží k uložení dat k zobrazení. Pro tento účel byla navržena třída *GraphMainBurst*, která slouží k uložení jedné dávky dat. Tato třída je univerzální pro uložení všech datových tříd, které obsahuje hlavní archiv. Konstruktoru se předávají čtyři hodnoty typu *float*, které se načtou z libovolné třídy hlavního archivu. Tato třída slouží pouze k uložení dat. K manipulaci s daty, jejich načítání a následnému předání grafu byla navržena třída *GraphMainBurstStore*.

GraphMainBurstStore představuje hlavní úložiště dat pro graf. Zároveň ale implementuje i metody k jejich načtení. Konstruktoru se předává cesta k souboru, pět proměnných typu *integer*, které obsahují číslo dat zvolených ve *spinneru*, konkrétně se jedná o U, I, THD, Fi a P. Jako poslední je konstruktoru předán datový typ *long* udávající posun, který byl vybrán jako časový interval mezi jednotlivými záznamy.

Pro čtení binárního souboru se vytvoří instance třídy *ArchiveReader*, na které je zavolána metoda *initDataInfo*, která inicializuje informace o datech, které soubor obsahuje. Před samotným načtením zvolených dat je potřeba otestovat vybraný posun



Obr. č.6 Hlavní graf zobrazuje křivky jednotlivých záznamů, napravo a nalevo jsou zobrazeny hodnoty. Je zobrazen pouze čas prvního a posledního zobrazeného záznamu z důvodů ušetření místa.

a následně zjistit, jestli se budou ukládat data pro jednu nebo dvě dávky. Protože při výběru intervalu zobrazovaných dat v předchozí aktivitě se nevytváří instance třídy *ArchiveReader* pro zjištění minimálního posunu, může být zvolený interval menší, než v jakém byla data zaznamenána. Tato skutečnost je otestována a případně je posun změněn tak, aby vyhovoval intervalu, v jakém byly záznamy pořízeny. Dále je otestováno, zdali se budou ukládat dvě dávky dat nebo pouze jedna. Test se provádí z důvodů ušetření paměti, protože by bylo nevýhodné inicializovat dvě pole pro obě dávky, když bude ukládána jen jedna. K tomuto účelu slouží právě těch pět proměnných typu *integer*, které předáváme konstruktoru. Poslední krokem před inicializací datových polí je zjištění jejich délky. Délka se určuje pomocí počtu záznamů, které vynásobíme intervalem, s jakým byly zaznamenány, a to celé vydělíme posunem, který byl třídě předán v konstruktoru.

Načítání probíhá v cyklu *for* a je nutné projít všechny záznamy. Pro načtení hlavního archivu je použita metoda *readMain* instance třídy *ArchiveReader*. V případě, že byl interval zvolen větší, než je interval pořízení záznamů, byla vytvořena proměnná *jump*. Tato proměnná obsahuje počet záznamů, které budou přeskočeny. Inicializuje se z posunu, který je vydělen intervalem pořízení záznamů. Přitom při každém projetí cyklu *for* je tato proměnná decrementována a v případě, že je rovna nule, se záznamy uloží a proměnná je opět inicializována. Pro načtení dat byl použit rozhodovací

algoritmus *switch*. Celkem obsahuje pět *switchů*, každý pro jednu hodnotu *integer* předanou v konstruktoru. Pro porovnání se používají statické proměnné, které jsou nadeklarovány v třídě *GraphMainContentChooser*, sloužící pro zvolení dat k zobrazení. Pro načtení pěti druhů dat byly vytvořeny metody *loadDataU, loadDataI, loadDataTHD, loadDataFi* a *loadDataP*. Tyto metody jsou identické, přebírají pozici v poli pro zapsání, data a datový typ *boolean*, určený pro zjištění, jestli má zapisovat do prvního, nebo druhé pole. Liší se pouze v datech, která přebírají. Po rozlišení pole a zapsání hodnot je ještě provedena kontrola minima a maxima. Minimum a maximum je důležité pro zobrazení dat. Pro kontrolu byly navrženy metody *getMax* a *getMin*. Ty jsou pro všechny metody načítající data identické, předává se jim aktuální maximum nebo minimum a pět hodnot typu *float*. Metody vrací hodnotu typu *float*, která vyjde z rozlišovacího algoritmu jako nejmenší či největší. Po projití všech *switchů* se ještě uloží čas aktuálně načtených záznamů do pole *longů*.

4.5.3 Třída pro vykreslení grafu

V androidu neexistuje žádný *widget* pro vykreslování grafů, a proto byla vytvořena třída dědící od objektu *View*. V této třídě je přepsána metoda *draw*, která přebírá instanci třídy *Canvas*. Metoda je volána při prvním zobrazení, a protože objektu v tomto okamžiku zatím nebyla předána data, tak je vykreslení grafu ošetřeno pomocí proměnné *boolean*. K získání dat byly navrženy metody setBurst1 a setBurst2. Obě mimo jiné aktualizují příslušnou proměnnou typu *boolean*, která ve volání metody *draw* zpřístupní vykreslení grafu. K správnému vykreslení je ještě potřeba inicializovat maxima a minima dosahované hodnotami předanými grafu. Pro tento účel byly navrženy čtyři metody *setMax* a *setMin*. Nakonec inicializace dat je ještě potřeba grafu předat čas prvního a posledního záznamu k zobrazení. Opětovné vykreslení *widgetu* již zavolá metody pro vykreslení, tzv. *initPaint, initAxis, drawChart* a *drawValueAxis*.

Metoda *initPaint* slouží k inicializaci instance třídy *Paint* a její základní nastavení. Dále jsou inicializovány proměnné obsahující informace o výšce a šířce *widgetu* a odsazení grafu od okrajů.

Metoda *initAxix* na rozdíl od předchozí provede zakreslení do grafu, konkrétně vykreslí postranní osy, na které budou později zapsány hodnoty. Metoda přebírá instanci třídy *Canvas*, právě třída *Canvas* umožňuje kreslit tvary.

Další volaná metoda, *drawChart*, vykreslí samotný graf. Metodě je předána instance třídy *Canvas* a dvě instance třídy List s datovým typem *GraphMainBurst*.

Jedná se o první a druhou dávku dat. Kreslení probíhá v cyklu *for* pomocí metod *joinValues1* a *joinValues2* pro první a druhou dávku. Metody jsou identické, jediný rozdíl je použití barev. Metodám se předává instance třídy *Canvas*, dvě instance třídy *graphMainBurst* a *float*, který slouží k určení posunu. Třída *graphMainBurst* obsahuje čtyři proměnné, proto je volána metoda *drawLine* čtyřikrát. Metodě *drawLine* se nepředávají přímo hodnoty, které obsahuje *graphMainBurst*. Hodnoty jsou upravené pomocí metody *getDrawValue*, která přebírá hodnotu *float*, minimum a maximum. Hodnota, kterou tato metoda vrací, je upravena ve správném poměru tak, aby nepřetekla displej ani z jedné strany a aby byla na displeji správně umístěna. Poslední metoda *drawValueAxis* slouží k zapsání hodnot na osy a času prvního a posledního záznamu. Přebírá pouze instanci třídy *Canvas*. Hodnoty k zapsání se určují z minima a maxima.

4.5.4 Aktivita pro zobrazení dat hlavního archivu

Pro zobrazení dat hlavního archivu byla vytvořena třída *GraphMainClass*, která dědí z třídy *View*. V XML layoutu pro zobrazení dat je pak pouze lineární layout obsahující právě tuto třídu pro hlavní graf. Po spuštění aktivity se nastaví layout, z kterého je získána instance třídy *GraphMainClass*. Té je nastaven *OnClickListener* sloužící pro zobrazení dialogu pro pohyb v grafu. Dále je vytvořena instance třídy *Display*, z které je získána výška displeje a vypočítán počet záznamů, který může být najednou zobrazen. Načtení hodnot z hlavního archivu probíhá v této třídě. Jedná se o časově náročnou operaci, proto je zobrazen *ProggressDialog* a je vytvořeno vlákno, které z *intentu* vytvářející tuto aktivitu získá potřebné hodnoty a předá je instanci třídy *GraphMainBurstStore*. V ní proběhne přečtení a uložení hodnot. Poté je z vlákna zavolána metoda *dismiss* na instanci *ProggressDialogu* a *handleru* odeslána prázdná zpráva informující pouze o úspěšném načtení, následně je zobrazen graf. Pokud by mělo být načtení neúspěšné, uživatel je o tom informován zprávou.

První zobrazení a inicializace dat jsou provedeny v metodě *handleMessage* instance *handleru* určeného k informování o načtení dat. Jsou provedena nastavení maxim a minim a také jednotek, ve kterých jsou předaná data uložena. Poté je zavolána metoda *setLastBurst*, která zobrazí dávku dat, které byly zaznamenány jako poslední. Kromě této metody byly navrženy i další umožňující pohyb v grafu. Jedná se o *setFirstBurst*, *setPreviousBurst*, *setNextBurst*, *setPrevious* a *setNext*. Tyto metody se rozlišují tím podle směru pohybu v grafu a počtu kroků. Nejdůležitější proměnné,

s kterými tyto metody pracují, jsou aktuální pozice, celkový počet záznamů a počet záznamů najednou zobrazených.

Metody *setLastBurst* a *setFirstBurst* neberou na současnou pozici ohled a nastaví ji buď na začátek nebo na konec grafu. Konec ale není poslední záznam, mezi posledním záznamem a koncem je mezera, která je ekvivalentní počtu zobrazitelných záznamů tak, aby byl displej vždy naplněn. Z toho vyplývá, že proměnná pozice nabývá hodnoty od nuly do počtu záznamů mínus počet záznamů, které je možné najednou zobrazit.

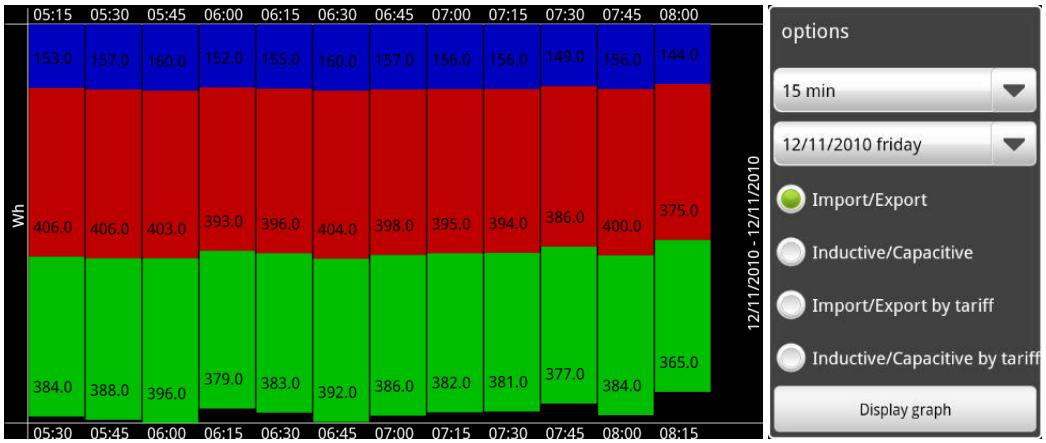
Metody *setPreviousBurst* a *setNextBurst* nastavují předchozí nebo další dávku. Dávka znamená, že posunou záznamy o počet, který se vejde na displaj. Změní všechny záznamy aktuálně zobrazené na další nebo předchozí část grafu. Metody nejsou typu *void*, vrací *true,* pokud je posun úspěšný, a *false,* pokud není možno posun provést. Posun není možné provést pouze v případě, že pozice plus počet zobrazitelných záznamů je větší jak počet všech záznamů, nebo v opačném případě menší než nula. V tomto případě je volána bud metoda *setFirstBurst* nebo *setLastBurst*. Pokud je posun možné provést, uskuteční se aktualizace pozice, vyčistí se *ArrayList* a je znovu naplněn požadovanými hodnotami.

Poslední dvě metody, *setPrevious* a *setNext,* jsou typu *void*. Jedna provede posun o jednotku dopředu a druhá dozadu. Před samotným posunem je testováno přetečení podle posunu, o který se jedná. Pokud podmínky vyhovují, provede se rotace v závislosti na metodě buď o jeden prvek doprava, nebo doleva. A prvek, který se přesunul na začátek nebo konec je nahrazen novým. Metody samozřejmě aktualizují pozici. Ve všech těchto metodách je implementováno rozlišení, jestli jsou aktivní obě dávky nebo pouze jedna.

Pro volání těchto metod byl navržen dialog, který se zobrazí po dotyku displeje na libovolném místě. Po kliknutí na tlačítka, která obsahuje, zůstává zobrazen a jeho zmizení je docíleno kliknutím mimo něj.

Dialog je navržen pomocí jednoduchého lineárního layoutu, ve kterém jsou pod sebou umístěná tlačítka. Tlačítkům je přiřazen obrázek, který uživatele informuje o typu posunu. Během zobrazení dialogu je na jeho pozadí vidět graf, a proto uživatel přesně ví, mezi kterými daty se pohybuje.

4.6 Zobrazení electricity grafu



Obr. č.7 a) *Electricity* graf zobrazující hodnoty formou sloupečků. Nalevo je zobrazena jednotka dat, napravo první a poslední datum zobrazených záznamů. Čas intervalu každého sloupečku je zobrazen nahoře a dole. b) Dialog výběru obsahu, ve kterém si může uživatel zvolit interval, data k zobrazení a den, který se má zobrazit jako první.

Hlavní graf byl řešen jako spojnicový, *electricity* graf zobrazuje hodnoty do sloupečků. Do jednoho sloupečku může být zobrazeno až šest hodnot, protože import se zobrazuje proti exportu a obě položky se skládají z tří hodnot. Na rozdíl od hlavního archivu se před zobrazením grafu nespouští žádná aktivita. *Electricity* graf má míň možností na výběr, a proto volba obsahu grafu byla vyřešena pouze pomocí dialogu.

Po spuštění aktivity se nastaví layout, ten obsahuje pouze objekt *GraphElmerClass*, který dědí z třídy *View*. Objekt se inicializuje a nastaví se mu *OnLongClickListener*, který slouží k vyvolání dialogu pro zvolení obsahu. Poté se z velikosti displeje získá počet zobrazitelných záznamů a z *Intentu* se vyvodí cesta k souboru, jehož data se budou zobrazovat. Zobrazení samotného grafu musí předcházet uživatelovo zvolení jeho obsahu, proto je pomocí metody *showDialog* zobrazen dialog pro výběr obsahu.

4.6.1 Dialog pro výběr obsahu

Layout, z kterého se dialog vytváří, obsahuje jeden lineární layout s vertikální orientací, ve kterém jsou umístěny jednotlivé ovládací prvky. Jde o dva *spinnery*, *RadioGroup*, která obsahuje objekty typu *RadioButton*. V jedné *RadioGroup* může být zvolen vždy pouze jeden *RadioButton*. A na závěr je v layoutu jedno potvrzovací tlačítko, které zobrazí graf.

Po jeho zobrazení dojde k inicializaci prvků a poté se volají metody *initTimeShift* a *initStartTime*. Metoda *initTimeShift* slouží pouze k inicializaci *spinneru*, který obsahuje časový posun, nic nevrací, ani žádné objekty nepřebírá.

Metoda *initStartTime* inicializuje druhý *spinner*, metoda přebírá hodnotu *integer*, která složí k nastavení intervalu pro vytvoření adaptéru. Nicméně hodnota je programově nastavena na jedna a nelze ji prozatím změnit. S tímto nastavením bude adaptér obsahovat den posledního záznamu a pak každý předchozí pátek. Čas, který vybereme ve druhém *spinneru*, udává poslední záznam, který se zobrazí. Po projití intervalu od posledního záznamu k prvnímu je adaptér naplněn a následně předán *spinneru*, který zobrazí příslušná data. Zároveň s adaptérem je vytvořen pomocný *ArrayList* obsahující časy adaptéru v milisekundách. Po úspěšné inicializaci *spinneru* se volá metoda *initShowButton*.

V metodě *initShowButton* se nastavuje *OnClickListener* tlačítku pro zobrazení grafu. Po kliknutí se vytvoří instance třídy *GraphElmerBurstStore*, která podobně jako v hlavním archivu má funkci úložiště záznamů a zároveň slouží k jejich načtení. Třídě se předává cesta k souboru, poslední čas k zobrazení a posun, obě tyto hodnoty jsou určeny zvolenou položkou ve *spinneru*. Třída okamžitě po inicializaci provede načtení záznamů, které není nutné provádět ve zvláštním vláknu, protože se na rozdíl od hlavního archivu nejedná o časově náročnou operaci. Pokud je načten potřebný počet záznamů k zobrazení, jsou grafu nastaveny hodnoty k zobrazení, které si uživatel volí pomocí *RadioButtonů*. Následně je zrušen dialog, jsou nastaveny maximální a minimální hodnoty a pomocí metody *getBurst* je získána dávka odpovídající velikosti displeje. Graf je připravený k zobrazení, posledním krokem je zavolání metody *invalidate*.

Pohyb v grafu je opět vyřešen pomocí dialogu obsahující tlačítka s příslušným obrázkem vypovídajícím o jeho funkci. Pohyb je vyřešený pomocí metod *getNextBurst*, *getPreviousBurst*, *getNext*, *getPrevious*, *getStartBurst* a *getEndBurst*. Tyto metody kontrolují, zdali je pohyb možný, a pokud je tomu tak, je nastavena očekávaná pozice. Metody pouze nastaví pozici, o samotné vytvoření dávky pro graf se volá metoda *createBurst*, které se předává pozice a vrací *ArrayList* obsahující data k zobrazení. Data stačí předat grafu a zavolat *invalidate*.

4.6.2 Úložiště dat electricity archivu

Stejně jako u hlavního archivu i u tohoto je nutno data po přečtení někam uložit. Pro čtení a uložení záznamů slouží třída *GraphElmerBurstStore*, která využívá pro uložení jednoho záznamu třídu *GraphElmerBurst*.

Stěžejní u tohoto archivu také je, jakým způsobem se záznamy ukládají. Neukládá se jeden konkrétní záznam, nýbrž rozdíl mezi dvěma. Interval určuje, které dva záznamy se budou navzájem odečítat.

Třída *GraphElmerBurstStore* v konstruktoru přebírá tři proměnné, první je cesta k souboru, který bude načten, druhá je startovní čas, ten udává, který čas by měl být zobrazen při prvním vykreslení grafu, a poslední proměnná je posun určující časový interval, který bude mezi dvěma záznamy, které se odečtou. Třída se inicializuje v konstruktoru, převezme proměnné a vytvoří instanci třídy *ArchiveReader*, která poskytne informace pro čtení souboru. Archivy se čtou všechny, ale ukládají se pouze ty, které zapadají do časového posunu testovaného pomocí proměnné, která je předána této třídě. Pokud test proběhne, tj. čas posledního správného záznamu plus posun se rovná času aktuálně přečteného záznamu, vytvoří se instance třídy *GraphElmerBurst*, která se přidá do *ArrayListu*, který obsahuje záznamy k zobrazení. Poté jsou testovány maximální hodnoty sloužící pro správné vykreslení grafu. Pro tuto událost je navržena metoda *checkMax*, které se předává součet trojice hodnot například importu a aktuální maximální hodnota importu. Funkce vrací maximum, které vznikne porovnáním předaných hodnot. Tato kontrola se provede pro všechny trojice hodnot.

Třída *GraphElmerBurst* je navržena tak, aby ukládala jednu dávku hodnot. Konstruktoru se předávají dvě instance třídy *SmpArchiveElmer* a instance třídy *SmpInstallConfig*. Hodnoty, které se používají při kreslení grafu, vzniknou odečtením novějšího záznamu *SmpArchiveElmer* od staršího a vynásobením multiplikační proměnou získanou metodou *getFinishValue* z instance třídy *SmpInstallConfig*. Kromě samotných hodnot je k záznamům uložen čas prvního a druhého záznamu jako *long*.

4.6.3 Třída pro vykreslení grafu

Electricity graf se vykresluje jako sloupcový. Ke kreslení grafu byla vytvořena třída *GraphElmerClass*, která dědí od třídy *View*. Kreslení probíhá přepsáním metody *draw*. Je nutné zkontrolovat pomocí *boolean* proměnné, zdali byla třídě předána data k zobrazení grafu. Pro získání dat byla vytvořena metoda *setBurst*, která přebírá instanci třídy List s datovým typem *GraphElmerBurst*. Dále je potřeba třídě předat maxima,

kterých jednotlivé hodnoty nabývají, a nakonec metodou *setValuesShow* nastavit, které hodnoty se mají zobrazit.

Samotné vykreslení probíhá podobně jako u hlavního archivu pomocí metod instance třídy *Canvas*. V metodě *draw* třídy *GraphElmerClass* se pomocí rozhodovacího algoritmu *switch* na základě předané hodnoty rozhodne, jestli se bude zobrazovat import a export, induktivita a kapacita, import a export podle tarifů nebo induktivita a kapacita podle tarifů. Kreslení probíhá stejně, jen se liší v zobrazených hodnotách. Po testu námětu zobrazení se nastaví proměnná pozice, na které se bude začínat. Následně je vypočítána proměnná *shift*, která určuje poměr pro přepočtení reálné hodnoty na hodnotu zobrazitelnou na displeji. K výpočtu se používá šířka displeje a maximální hodnoty, které se budou zobrazovat. Kreslení grafu může začít. K tomuto účelu slouží algoritmus *for*, který projede celou dávku získanou metodou *setBurst*. Na displeji z úsporných důvodu u každé hodnoty není datum, pouze časy intervalů, ve kterém se záznamy odečítaly, první čas nad grafem a druhý pod grafem. Na kraji displeje je potom vypsáno datum prvního a posledního zobrazeného záznamu, výpis je ošetřený podmínkou *if*, která kontroluje pozici v dávce a při čtení prvního a posledního záznamu proběhne vykreslení hodnot. Při načítání hodnot z instance třídy *GraphElmerBurst* probíhá jejich přepočtení pomocí metody *drawValues*, které se předává hodnota a proměnná *shift*. Tyto dvě proměnné jsou vynásobeny a vráceny jako *float*. Hodnoty k zobrazení jsou načteny, a proto je možné začít vykreslovat sloupečky. Sloupečky jsou vykreslovány v algoritmu *for* pomocí metody *drawLine*, která se opakuje 70krát pro každou ze tří hodnot, čímž vznikne sloupeček o třech barvách. V průběhu vykreslování je pravidelně inkrementována pozice. Po vykreslení sloupečků je ještě nutné zapsat ke každé barvě hodnotu, která byla aktuálně vykreslena, a může být načtena další instance třídy *GraphElmerBurst,* tj. hlavní cyklus se opakuje.

5. Vyhodnocení

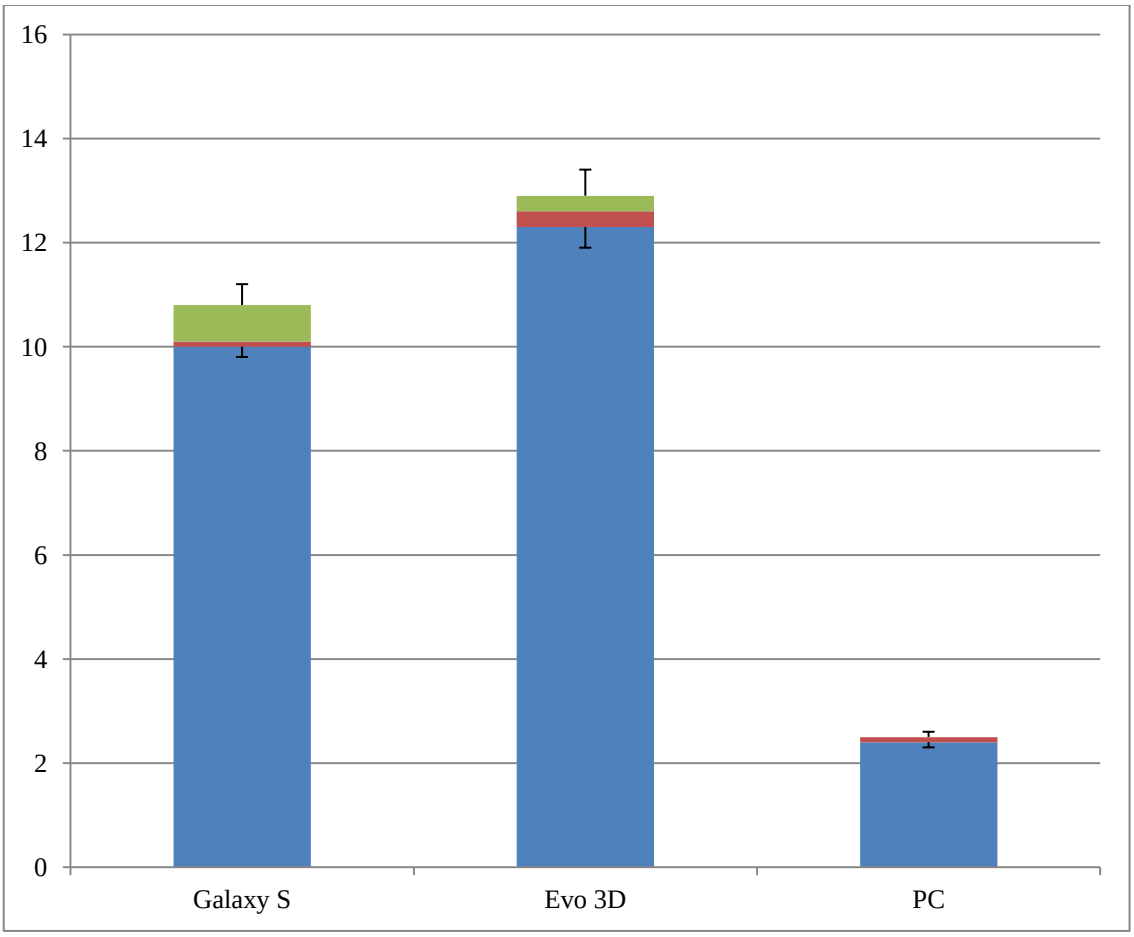
Po úspěšné implementaci programu byly provedeny testy, ve kterých byly zkoumány rozdíly v časech při stahování, rozbalování a zobrazování CEA archivů na notebooku a mobilním zařízení.

Testy na notebooku proběhly v programu Envis. Na notebooku byl nainstalován Windows 7 Home Premium, jednalo se o 64bitovou verzi systému. Notebook byl osazen procesorem Intel Core i5 M460, který tikal na frekvenci 2.53GHz a byl podporován 4GB operační paměti.

Jako jedno z mobilních zařízení byl použit výkonný stroj od firmy *HTC*, model *EVO 3D* s operačním systémem Android 2.3.4. V útrobách tohoto zařízení se nachází procesor *Qualcomm MSM8260*, dvoujádrový procesor tikající na frekvenci 1.2GHz, kterému asistuje 1GB operační paměti RAM.

Druhé mobilní zařízení zastoupil *Samsung Galaxy S,* využívající systém Android 2.3.3. Mobilní zařízení je poháněné procesorem *Samsung S5PC110*, jedná se o upravený *Cortex-A8,* tikající na frekvenci 1GHz. Procesoru asistuje 512MB paměti RAM. Zajímavostí tohoto modelu je, že nemá slot pro paměťové karty, protože je od výrobce vybaven vnitřní pamětí 8GB.

5.1 Stahování CEA archivu



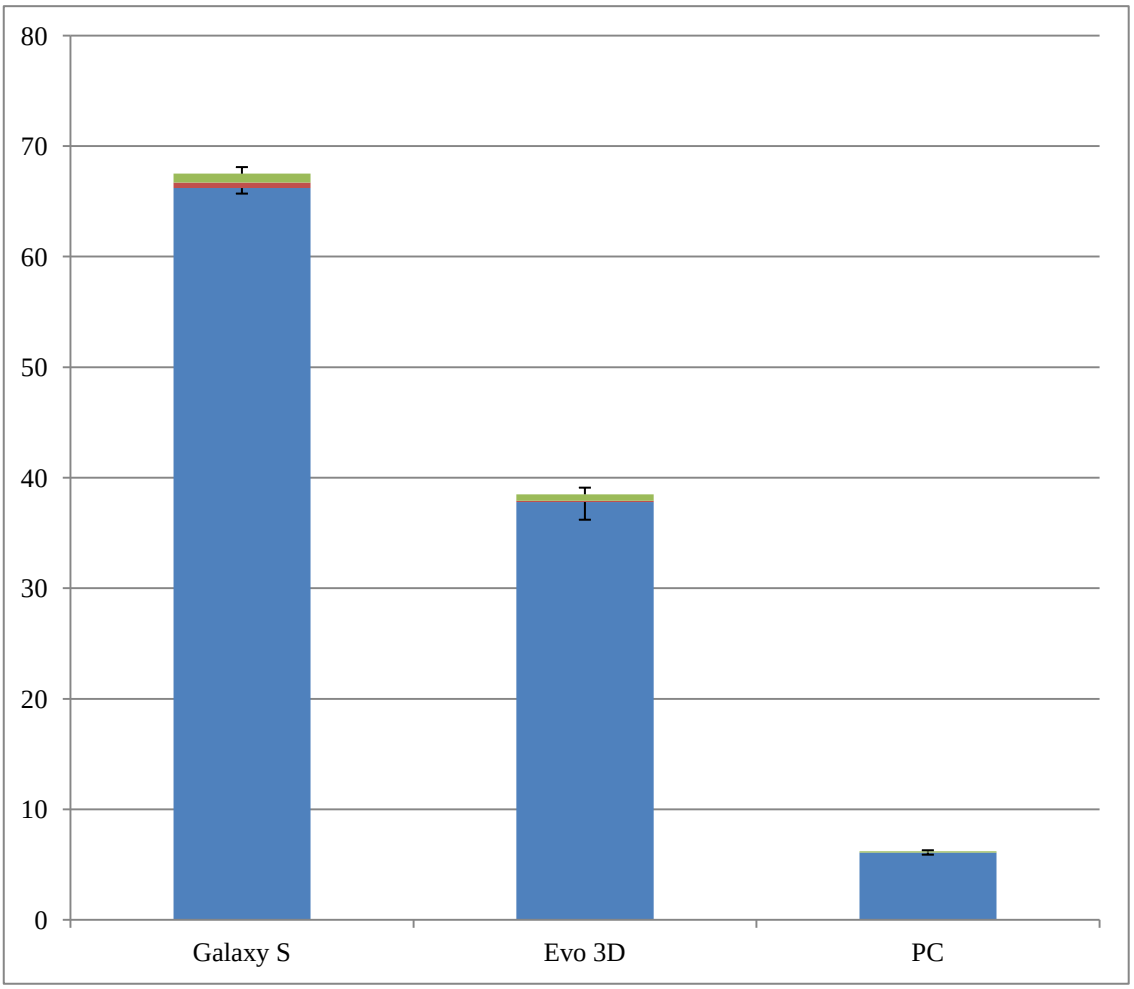
Obr. č.8 Porovnání rychlosti stahování CEA archivu.

Pro testování stažení a vytvoření CEA archivu byl použit analyzátor, který měřil nulové hodnoty, respektive nebyl připojený k síti, proto pro všechny hodnoty zapsal nulu. Pro změření rychlosti stažení toto není stěžejní, protože všechny zařízení měly stejné podmínky. K analyzátoru byly zařízení připojené pomocí *WiFi* sítě, tím pádem nebylo ani jedno zařízení omezené datovým tarifem.

Stahovaný CEA archiv obsahoval hlavní, *electricity* a log archiv. Časový interval byl nastaven na jednu hodinu.

Notebook má jasný náskok, který by, při stahování delšího časového intervalu, ještě vzrostl, a nechal mobilní zařízení daleko za sebou. Zajímavé je, že *Evo 3D* zaostává za *Galaxy S* i přesto, že má vyšší výpočetní výkon. Je to způsobené faktem, že *Samsung* má pevnou vnitřní paměť s rychlejším přístupem. Z tohoto zjištění logicky vyplívá, že omezení mobilních zařízení není v jejich výkonu, nebo rychlosti datového připojení, nýbrž v rychlosti zápisu do paměti.

5.2 Rozbalování CEA archivu



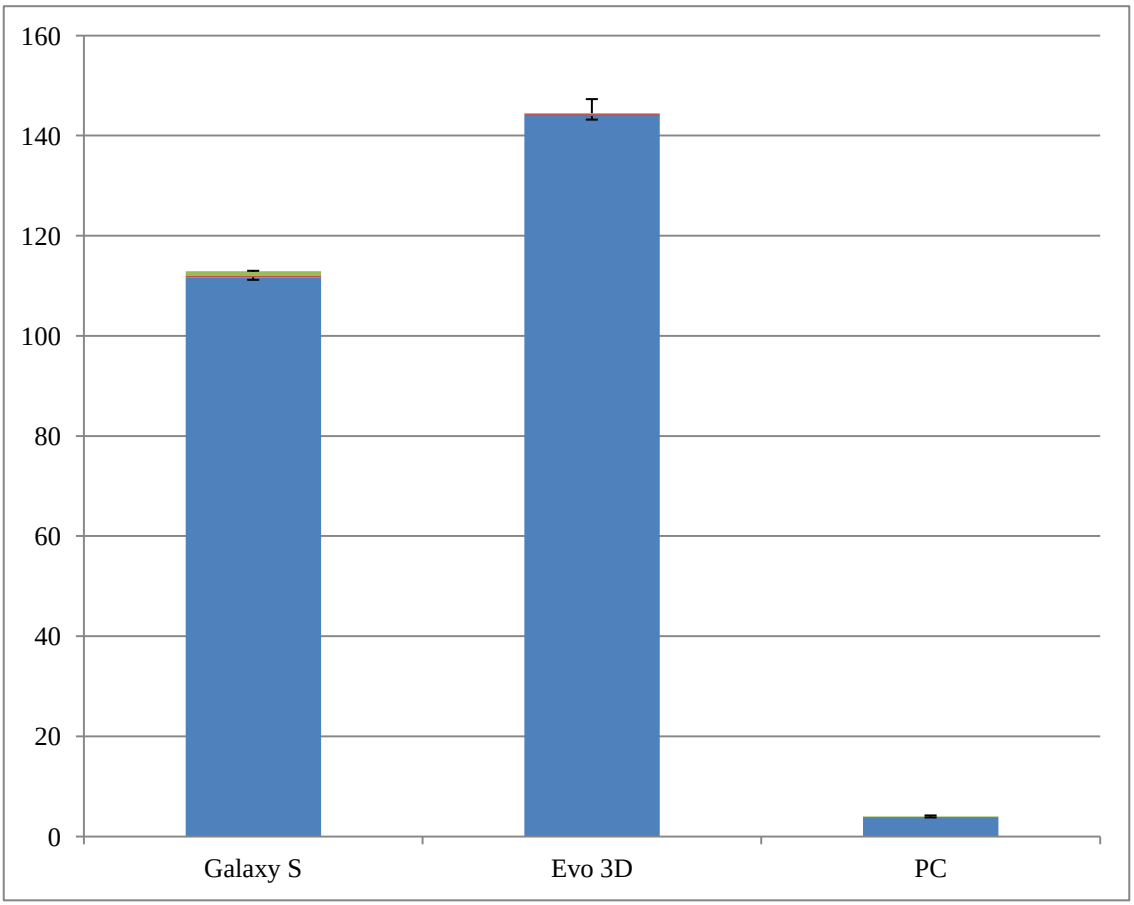
Obr. č.9 Porovnání rychlosti rozbalování CEA archivu.

Rozbalování archivů bylo testováno na dvou zabalených souborech. První měl velikost 20MB a obsahoval sedm binárních souborů a druhý měl velikost 79MB a obsahoval deset binárních souborů.

Ztráta mobilního telefonu nebyla příliš velká. Mobilní zařízení rozbaluje archiv přibližně 5krát pomaleji. V grafu je zohledněna i skutečnost, že ENVIS po rozbalení souboru ještě chvíli zpracovává data.

U rozbalování archivu hrál hlavní roli výkon, rychlost čtení a zápisu do paměti byla vedlejší. Z grafu je jasně vidět, že *Evo 3D* je téměř dvakrát rychlejší, což zhruba odpovídá jednomu jádru navíc.

5.3 Zobrazování binárních souborů



Obr. č.10 Porovnání rychlosti zobrazení hlavního grafu.

Při zobrazení grafů byly použity binární soubory, které byly získány při testování rychlosti rozbalování. Test byl proveden na *electricity* a hlavním binárním souboru.

Při zobrazování je rozdíl mezi výkonem mobilních zařízení a notebooku mnohem znatelnější, ale pouze v případu hlavního archivu. *Electricity* archiv obsahuje oproti hlavnímu mnohonásobně méně dat, proto se rozdíl ve výkonu neprojevil a graf byl zobrazen prakticky okamžitě u všech zařízení, z tohoto důvodu nebyl archiv zanesen do porovnávacího grafu. U hlavního archivu už tomu bylo jinak, v případě většího souboru bylo načítáno 172 646 záznamů. Mobilní zařízení díky tomuto počtu bylo značně zatíženo a i přes veškeré optimalizace, které se týkají načítání pouze určitých dat, si aplikace v operační paměti zabrala 22MB. Z grafu je vidět, že výpočetní výkon šel opět na druhou kolej a hlavní roli hrála rychlost přístupu k paměti.

6. Závěr

V práci jsou vyřešeny základní operace pro správu, přenos a částečné zobrazení dat z analyzátoru.

Podařilo se úspěšně implementovat stažení CEA souboru a jeho rozbalení, popsané v kapitolách 4.2 Aktivita pro stažení CEA souboru a 4.3 Aktivita pro rozbalení CEA souboru, i přes problémy Javy s rozbalováním archivů obsahující soubory, které mají v názvu diakritiku.

A nakonec i zobrazení grafů. Hlavního grafu, jímž se zabývá kapitola 4.5 Zobrazení hlavního grafu. A *electricity* grafu, jehož řešení je k nalezení v kapitole 4.6 Zobrazení *electricity* grafu. Zobrazení grafů společně i s jinými operacemi rozhodně není výkonnostně nenáročné, bylo třeba počítat s omezeným výpočetním výkonem a operační pamětí mobilního zařízení. Program proto bylo nutné co nejvíce optimalizovat a navrhnout tak, aby nebyl náročný na paměť RAM. To se týká hlavně zobrazování grafu hlavního archivu, ve kterém jeden záznam obsahuje několik desítek hodnot typu *float*. I přesto bylo zobrazení optimalizováno tak, aby na mobilním zařízení probíhalo i v případě, že se načítá přes 150 000 záznamů.

Co se týče správy stažených CEA souborů a rozbalených binárních archivů, tak ta je řešena v kapitolách 4.3 Aktivita pro rozbalení CEA souboru a 4.4 Aktivita pro práci s binárními archivy. Pro tyto aktivity byl navržen jednotný layout. Aktivity se liší ve funkčnosti, jedna soubory rozbaluje a druhá zobrazuje grafy. U binárních archivů je navíc možné zobrazit CEA informace, které zobrazují informace z hlediska CEA archivu.

Prostor pro rozšiřování programu rozhodně existuje, je potřeba dokončit vizualizaci pro zbylé binární soubory, a pokusit se o další optimalizace, které by urychlily chod aplikace především během stahování a vytváření CEA souborů z analyzátoru.

Hlavní myšlenkou tohoto projektu byla možnost uživatele s mobilním zařízením přijít k analyzátoru, který je připojený například k wifi síti, a pomocí tohoto programu se k němu připojit a stáhnout data, která ho zajímají. Mobilní zařízení poté stažená data zapíše do binárních souborů a zabalí do CEA archivu, čitelného pro program ENVIS. S těmito staženými daty poté uživatel připojí mobilní telefon k notebooku a konkrétní data si z paměti telefonu stáhne a zobrazí. Další funkcí mobilního zařízení je možnost zobrazit data *electricity* a *main* souboru přímo v mobilu.

Literatura

[1] What is Android?. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[2] LinearLayout. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/LinearLayout.html

[3] TextView. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/TextView.html

[4] GridView. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/GridView.html

[5] Spinner. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/Spinner.html

[6] ProgressDialog. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/app/ProgressDialog.html

[7] ListView. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/ListView.html

[8] TimePicker. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/TimePicker.html

[9] DatePicker. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/widget/DatePicker.html

[10] Context. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/content/Context.html

[11] Intent. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/content/Intent.html

[12] Environment. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/os/Environment.html

[13] Handler. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/os/Handler.html

[14] Calendar. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/java/util/Calendar.html

[15] ZipOutputStream. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/java/util/zip/ZipOutputStream.html

[16] ContextMenu. GOOGLE. *Android developers* [online]. © 2012 [cit. 2012-04-29]. Dostupné z: http://developer.android.com/reference/android/view/ContextMenu.html

[17] KMB Systems: Přístroje SMV/SMP/SMPQ - Návod k obsluze.

[18] KMB Systems: SMV/SMP/SMPQ *Multifunctional Panel Meters & Power Quality Analyzers Communication Protocol Manual*, 2009.

[19] KMB Systems: *Algorithm for downloading data and creating CEA files for ENVIS,* 16. září 2011.

[20] *Java* [online]. ©1995, 2010 [cit. 2012-05-16]. Dostupné z: http://www.java.com/en/

[21] *Eclipse - The Eclipse Foundation open source community website* [online]. © 2012 [cit. 2012-05-16]. Dostupné z: http://www.eclipse.org/

[22] MURPHY, Mark L. *Android 2*: *průvodce programováním mobilních aplikací*. Vyd. 1. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.